Simulation Modelling Practice and Theory 19 (2011) 2206–2225

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

Profile-based spatial partitioning for parallel simulation of large-scale wildfires

Song Guo [*], Xiaolin Hu

Department of Computer Science, Georgia State University, Atlanta, GA 30303, USA

ARTICLE INFO

Article history:
Received 29 July 2010
Received in revised form 9 July 2011
Accepted 14 July 2011

Keywords:
Parallel and distributed discrete event simulation
Load balance
Task partition
Performance measurement
Large scale simulation

ABSTRACT

Spatial partitioning is commonly used for parallel simulation of spatial–temporal systems, such as simulations of wildfires. Achieving effective spatial partitioning is a challenging task due to the dynamic behavior of the simulation models. This paper presents a partitioning method named profile-based spatial partitioning for parallel simulation of large scale wildfires. The profile-based partitioning exploits the dynamic behavior of a wildfire spread simulation model and uses it as a profile to guide the spatial partitioning for parallel simulations. Experimental results show that the profile-based partitioning can increase the degree of parallelism and improve the scalability of parallel simulations of large-scale wildfires.

1. Introduction

Recent years have witnessed an increasing number of wildfires with increased burned areas as well as damages to the environment. Understanding the behavior of wildfires and predicting the fire intensity, rate of spread, and spread direction can support better management and control of wildfires. Towards this goal, several wildfire spread simulation models have been developed, including FARSITE [15], BehavePlus [14], Hfire [34], and DEVS-FIRE [9,12]. Simulating large-scale wildfires is a computation-challenging task because of the complexity of the fire spread model and the large size and long duration of the simulations. To improve the performance of wildfire simulations, parallel simulation techniques are needed.

To support parallel simulations we need to divide the simulation tasks and assign them to multiple processing units (PUs). This step of task partitioning is critical and can have significant impact on the performance of a parallel simulation. Wildfire simulation models are characterized by their dynamic behavior in both space and time. Due to the spatial–temporal behavior, spatial partitioning is commonly employed for parallel simulations of wildfires. In spatial partitioning, the space is divided into multiple regions, which are distributed to multiple PUs. When a fire spreads to a new region, it triggers the computation of the PU in charge of that region. This paper considers spatial partitioning for parallel simulation of wildfire spread using the DEVS-FIRE model. DEVS-FIRE is a discrete event wildfire spread and suppression model based on the DEVS formalism [2]. It employs a two dimensional cellular space model composed from multiple individual cells to represent the wildfire area. When simulating large-scale wildfires modeled by a large number of cells, there is a need to improve the simulation performance using parallel simulation techniques.

An effective spatial partitioning for parallel simulation of wildfires should increase the degree of parallelism and improve the scalability as well. For the DEVS-FIRE model, one approach of spatial partitioning is to divide the cell space into several

* Corresponding author.
E-mail addresses: sguo3@student.gsu.edu (S. Guo), xhu@cs.gsu.edu (X. Hu).

1569-190X/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.simpat.2011.07.003

size-equivalent sub cell spaces corresponding to the number of PUs, and then allocate each sub cell space (also called a partition) to a PU for parallel simulation [22]. However, this approach is ineffective because the execution sequence of the simulation tasks heavily depends on the fire spreading behavior, which is influenced by many factors such as ignition point, vegetation, terrain, and weather [23]. The dynamic fire spreading behavior can cause different PUs to have significantly different computation load at a given time. To provide a simple example, let us consider a simulation where a wildfire is ignited at a location far from the partitioning boundary. In this case, it will take a long time for the fire to spread to other partitions to trigger them to start the simulation. As a result, in the beginning of the simulation only one PU containing the ignition point engages in computation; while others are idle waiting for the fire to spread to their partitions. This type of temporally unbalanced load of computation degrades the simulation performance.

In order to overcome the problem mentioned above, this paper presents a profile-based spatial partitioning method to improve the performance of large scale simulation of wildfires. The profile-based spatial partitioning is motivated from the observation of fire spreading behavior in DEVS-FIRE: the simulation focuses on the burning frontline of the fire that is the active area; no computation is needed for the area that is not ignited or already burned out. Thus if all PUs can concentrate on the active area of the fire, the time of waiting will be greatly reduced so that better parallelism can be achieved. Motivated by this, the profile-based partitioning method exploits the model behavior, i.e., the spatial–temporal wildfire spreading behavior, and uses it as a profile for guiding the task partitioning for parallel simulation. In our work, the behavior profile is obtained from a simulation using low resolution GIS data. The ignition sequence of the cells from the low resolution simulation represents a behavior profile of the high resolution simulation. This profile is used to partition the cell space for parallel simulation of DEVS-FIRE. A low resolution simulation runs much faster than the high resolution simulation. As will be shown later, the time spent in generating the profile (using the low resolution simulation) can be easily compensated by the performance gains from the profile-based partitioning. In this paper, the parallel simulation algorithm that we employed is based on the time warp algorithm [10], which is an optimistic approach of time management [26] that uses a rollback mechanism to recover violations of local causality in order to maintain the local causality constraint. We note that an alternative approach of parallel simulations is based on conservative algorithms (see e.g., [6]). Conservative algorithms rely on the concept of lookahead in order to achieve concurrent processing of events. The wildfire simulation model considered in this paper does not have good lookahead properties because its fire spreading behavior depends on the dynamic changing wind speed/direction as well as the non-uniform slope, aspect, and fuel model of the forest cells. Also note that in this paper, we only consider static partitioning where the partitions are divided and assigned to PUs before a simulation starts.

The remainder of this paper is organized as follows. Section 2 reviews the related work in task partitioning. Section 3 gives an overview of the DEVS-FIRE model. Section 4 presents the profile-based spatial partitioning method for parallel simulation of large scale wildfire. Section 5 gives a theoretical performance analysis of the profile-based method. Section 6 provides experiment results and Section 7 concludes this work.

2. Related work

Parallel simulation has long been studied and many algorithms and optimizations techniques are developed. P-DEVS [18] is a DEVS-based formalism for the specification of complex concurrent systems organized as an interconnection of atomic and coupled interacting models. It provides means of handling simultaneous scheduled events, while keeping all the major properties of the standard DEVS. Since P-DEVS eliminates serialization constraints, it enables improved execution of models in parallel and distributed environments. The work of [30] describes the Aurora2 architecture and issues pertinent to parallel discrete event simulation (PDES) [7] executions in a desktop grid environment that must be addressed when distributing back-end services across multiple machines. This improved architecture allows large scale PDES applications to take advantage of the master/worker style of parallel workload distribution and execution across desktop grids and public resource computing infrastructures. In [33], the authors developed a model interoperability layer in the PDES environment by conforming to its event-object paradigm. The model interoperability layer uses a proxy-based approach to enable object models in a PDES to "publish" and "subscribe" their attributes similar to mechanisms in high level architecture (HLA) [1]. Their design is especially optimized for executing in cluster-like environment. Experimental results show that the optimization improves Data Distribution Management performance up to 12 times in a cluster environment compared with the unoptimized design. The work of [28] presents a new technique that combines multi-dimensional parallelism coherently for efficient parallel simulation of large-scale DEVS-based models on the IBM Cell processor, which has one Power Processing Element (PPE) and eight Synergistic Processing Elements (SPEs). The proposed technique tries to explore multi-dimensional parallelism to overcome the bottlenecks in simulations. The underlying design methodology is illustrated with detailed simulation profiles of two example Cell-DEVS models, which simulate wildfire propagation and flooding scenarios respectively. The developed algorithms attained overall speedups up to 70.6 and 83.32 in the fire and flood simulations respectively over the baseline implementation on PPE by using parallelized FC Synchronization Kernel (FSK) on 8 SPEs. In [5], Cicirelli et al. proposed an agent infrastructure (Theatre) for distributed simulations based on Java over high-level architecture/RunTime Infrastructure (HLA/RTI). The basic building blocks of their architecture are actors (agents) which have a public message interface and encapsulate a state of local variables and a behavior patterned as finite state machine. Theatres, which are mapped onto HLA federates, work as the execution platforms for actors. Actors can migrate between theatres to achieve load-balancing. They demonstrated an application to a complex simulation model based on unmanned aerial vehicles (UAVs).

Task partitioning is a critical step in parallel simulations. An effective partitioning is pivotal to enable parallel simulations to achieve high degree of parallelism. Himmelspach et al. [24] introduce a simulation algorithm and present partitioning and load balancing techniques that are tailored to the efficient distributed execution of PDEVS. They base their elaborations on the idea of minimizing interprocessor communication, which is a major bottleneck in distributed PDEVS simulation. Hershberger et al. [8] proposed a scheme for a stream of multidimensional points. This scheme provides a general-purpose summary of the spatial distribution of the points and can answer a wide variety of queries. In particular, their adaptive spatial partitioning method can be used to solve spatial versions of several classical data stream problems, including hot spots and approximate range counting. Nandy and Loucks [29] presented a static partitioning algorithm for conservative parallel logic simulation. The algorithm attempts to minimize the communication overhead and to uniformly distribute the execution load among the processors. It starts with an initial random partition and then iteratively moves processes between clusters until no improvements can be found (a local optimum). In [3], a simple partitioning scheme based upon the pair wise exchange was developed by Boukerche to reduce the overhead of the rollbacks of a hybrid parallel simulation of wireless networks. The synchronization protocol makes use of both conservative and time-warp paradigms. Sislak et al. in [13] present a distributed architecture for situated large-scale agent-based simulations with predominately local interactions. They spatially partitioned the simulated virtual environment and allocated a dedicated processing core to the environment simulation within each partition. The dynamic load-balancing is also integrated into the work so that their partitioning can greatly improve the scalability of the simulation platform. In their work, they used their approach to extend the AGENTFLY air-traffic test-bed. Thorough evaluation of the system is performed which confirms the improvement in scalability.

The agent-based community also makes great efforts and contributes for the research of the distributed simulation of large-scale models. Logan and Theodoropoulos [27] developed a framework for the distributed simulation of Multi-Agent Systems (MAS). Their systems employ the notion of "spheres of influence" as a basis to dynamically partition the shared state of the simulation model into logical processes. They define the "sphere of influence" of an event as the set of state variables read or updated as a consequence of the event. An algorithm is sketched for dynamically partitioning the simulation to perform load balancing. Chen et al. [4] proposed two routing approaches to access the shared states: the address-based approach, which locates data according to their address information, and the range-based approach, whose operation is based on looking up attribute value range information along the paths to the destinations. The two approaches are discussed and analyzed in the context of the systems in [27]. Suryanarayanan et al. [31] present a system for performing logical-time synchronized Range Queries over data in the context of parallel and distributed simulations of MAS by extending [4]. The design is evaluated from the perspective of the volatility of the data structure under different query and update loads and its performance in terms of service latency and execution time. A strong correlation between the data structure's volatility and its performance as a simulation kernel suggests that reducing the volatility has potential improvement over the system. There are contributions from others, too.

Spatial resolution is an important concept in spatial–temporal simulations, such as simulations of wildfires. In [25], Hu and Ntaimo develop a dynamic multi-resolution cellular space model for forest fire simulation. In their work, the change of cells' spatial resolutions happens dynamically and adaptively as the simulation proceeds. Buss and Ahner in [17] present a low-resolution, constructive entity-level simulation framework called Dynamic Allocation of Fires and Sensors (DAFS), which can rapidly configure, execute and complement complex high-resolution combat models. It allows an analyst to very quickly create a simulation model that captures the first-order effects of a scenario. DAFS can be used to explore large areas of the parameter space and identify interesting regions where high-resolution models can provide more detailed information. In the work of [19], the authors propose high-dimensional data clustering as a key interfacing component between simulation modules with different resolutions and use unsupervised learning schemes to recover the patterns for high-resolution simulation results. In [20], the authors present a multi-resolution simulation methodology which uses numerical optimization as a tool for maintaining external consistency between models of the same phenomena operating at different levels of temporal and/or spatial resolution.

3. DEVS-FIRE overview

Before describing the profile-based partitioning method, we give an overview of the DEVS-FIRE model this work is based on. Our overview focuses on the aspect that is related to the profile-based method presented in this paper. Other aspects and more details of the DEVS-FIRE model can be found in [12].

DEVS-FIRE is an integrated simulation environment for surface wildfire spread and containment simulation based on the Discrete Event System Specification (DEVS) formalism. It employs a two dimensional cell space to model the field where the fire occurs. The cells in the cell space are rectangular, whose sizes depend on the resolution of the GIS data (fuel, slope and aspect) [16]. The cell space contains individual forest cells each of which contains the GIS data and weather conditions assumed to be uniform within the cell. Each cell in the cell space is represented as a DEVS atomic model in the simulation and is coupled with its eight neighbor cells located at the N, NW, W, SW, S, SE, E, and NE directions respectively, as shown in Fig. 1. Consequently, the forest cell space is a coupled model composed of a number of coupled forest cell models. Fire spreads from one cell to another is enabled via message passing between the two cells. As a result, the fire spreading is

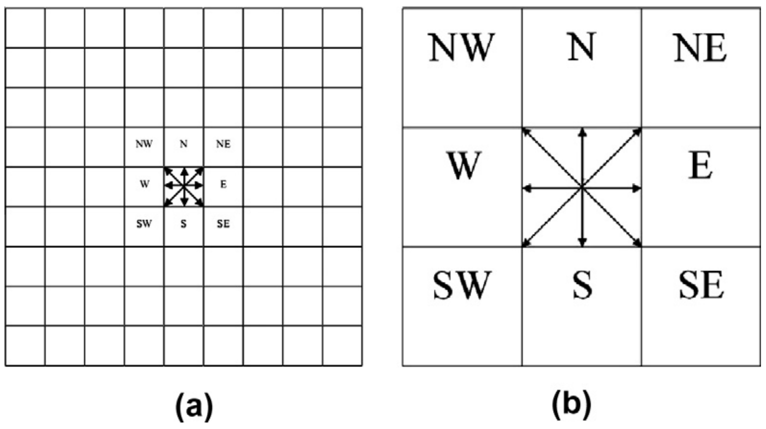


Fig. 1. Examples of 2D cell space: (a) 9 × 9 high resolution cell space with smaller cells (b) 3 × 3 low resolution cell space with larger cells.

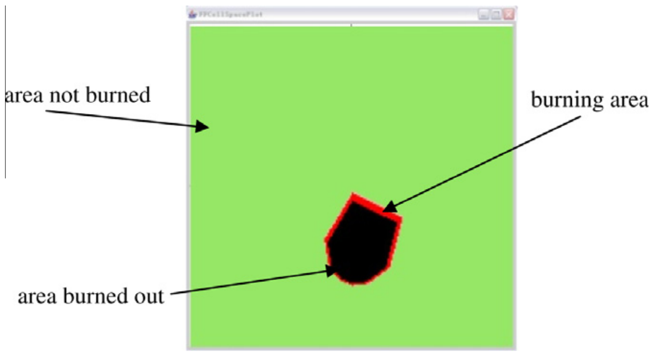


Fig. 2. A simulation scenario in DEVS-FIRE.

modeled as a propagation process where burning cells ignite their unburned neighbor cells. Fig. 1 shows two cell space models at different spatial resolutions for the same forest area. Fig. 1a shows a 9 × 9 cell space model and Fig. 1b shows a 3 × 3 cell space model, where each cell corresponds to 3 × 3 cells in Fig. 1a. The arrows in Fig. 1 represent the output couplings from the center cell to its eight neighbors.

In DEVS-FIRE, all cells are initially set to the unburned (passive) state. If a cell receives an ignition message and its fire line intensity is larger than its burning threshold, its state changes to burning. A cell, once ignited, calculates its rate of spread and spread direction using Rothermel's fire behavior model [16] based on its fuel, slope, aspect, and weather data. The rate (and direction) of spread is then decomposed into eight directions corresponding to its eight neighbor cells (see [12,32] for more details). Based on these rates of spread and the center-to-center distances between the cell and its neighbors, the cell schedules to ignite its neighbors (through message passing) according to the corresponding time delays. Finally, a burning cell changes to the burned state after it is burned out. From the simulation point of view, a cell is active when it is in the burning state. When a cell is active, the simulation needs to compute the cell's fire spread speed and direction, and to schedule ignition events for this cell to ignite its neighbors. On the other hand, if a cell has not been ignited or it is burned out, there is no computation needed for the cell. Fig. 2 displays a simulation scenario, where the unburned, burning, and burned out areas are marked. As illustrated in Fig. 2, in wildfire spread simulations all the burning cells form one or multiple spatially active areas, where intensive computation occurs. If we can concentrate all the computing power only on these active areas, it will accelerate the performance of large scale simulations. This motivates us to develop the profile-based partitioning method for parallel simulations of DEVS-FIRE.

4. Profile-based spatial partitioning for parallel simulation of large-scale wildfires

Based on the active area concept described above, we propose a profile-based spatial partitioning method for parallel simulation of large scale wildfire using DEVS-FIRE. The basic idea is to employ the result from a low resolution simulation as a profile of the fire spreading behavior to guide the partitioning process of the cell space among available PUs. The profile is an ignition sequence of the low resolution cells in the low resolution simulation. Although not as precise as the high resolution simulation, it contains knowledge regarding how the active area evolves in the high simulation. Once we have this profile,

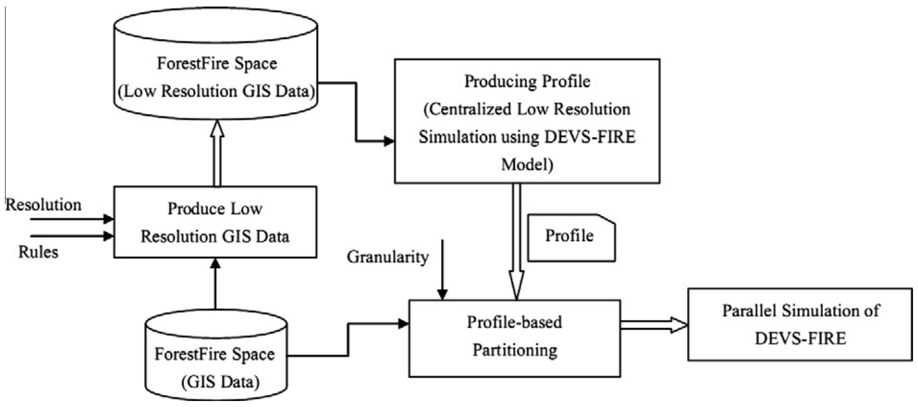

ForestFire Space (Low Resolution GIS Data)
Producing Profile (Centralized Low Resolution Simulation using DEVS-FIRE Model)
Resolution
Rules
Produce Low Resolution GIS Data
Profile
Granularity
Profile-based Partitioning
Parallel Simulation of DEVS-FIRE
ForestFire Space (GIS Data)

Fig. 3. Architecture of the profile-based spatial partitioning method.

we partition the cells among participating PUs so that all PUs are devoted to the predicted active area at an early stage of the parallel simulation. Fig. 3 shows the architecture of the profile-based spatial partitioning method. The architecture includes four parts that represent four major steps of the profile-based spatial partitioning method: (1) producing low resolution GIS data, (2) producing profile, (3) profile-based partitioning, and (4) parallel simulation of DEVS-FIRE based on the partitioning. Below we describe these four components in detail.

4.1. Producing low resolution GIS data

In order to obtain the profile from the low resolution simulation, we need a way to generate low resolution GIS data. Different resolution GIS data can be generated using dedicated software from the source files, e.g., satellite images. However this approach needs to have access to the source files and to use dedicated software. In our work, we produce low resolution GIS data from the original GIS data that already exist. We define R as the resolution factor from the original GIS data to the low resolution GIS data. This resolution factor specifies how many cells (in both horizontal and vertical dimensions) in the original GIS data are converted into one cell in the low resolution GIS data. For example, when $R = 3$, every 3×3 cells in the original GIS data are converted into one cell in the low resolution GIS data (see Fig. 1 for an illustration).

There are three types of GIS data used in DEVS-FIRE: fuel, aspect and slope. Different types of GIS data require different rules to transfer from high resolution to low resolution. In this paper, we follow the idea "dominant fuel, averaged aspect and slope" to generate the low resolution GIS data. We note that although these rules are arbitrary defined, they serve the purpose in our work to enable low resolution simulation for generating a profile of fire spreading behavior.

The idea of "dominant fuel" is to select the fuel type that has the maximum number of occurrence within an $R \times R$ partition in the high resolution cell space as the fuel type of the corresponding cell in the low resolution cell space, where R is the resolution factor. When there are several fuel types that have the equal maximum occurrence within the partition, we randomly choose one of them as the fuel type of the corresponding cell in the low resolution cell space. The idea of "averaged aspect and slope" is to use the average value of all the aspect/slope within the $R \times R$ partition as the slope or aspect of the corresponding cell in the low resolution cell space. Fig. 4 illustrates an example of this rule with $R = 4$. In Fig. 4a, the numbers represent the fuel types. On the left of Fig. 4a, there are nine occurrences of fuel type 7 in the 4×4 sub cell space. So fuel 7 is chosen to be the fuel type in the corresponding cell of the low resolution GIS data. On the right, each fuel type has 4 occurrences so a randomly chosen fuel type, in this case fuel type 9, is used as the fuel type in the corresponding cell of the low resolution GIS data. In Fig. 4b, the numbers on the left represent slope data and the numbers on the right represent aspects. On the left of Fig. 4b, the slope sum of 16 cells is 100. So the average slope 6.25 is calculated to be the slope of the corresponding cell in the low resolution GIS data. The aspect is calculated in a similar manner.

Fig. 5 shows an example of producing low resolution GIS data (with resolution factor $R = 4$) and the simulation results using the high resolution and low resolution GIS data. The low resolution GIS data are displayed in Fig. 5b, which are generated from a set of high resolution GIS data displayed in Fig. 5a. In Fig. 5a, the size of the cell space is 200×200 and the size of each cell is $2.5\ \text{m} \times 2.5\ \text{m}$. In Fig. 5b, the size of the cell space is 50×50 and the size of each cell is $10\ \text{m} \times 10\ \text{m}$. From the figures we can see that the "dominant fuel, averaged aspect and slope" rule is effective in that the low resolution GIS data conforms to the high resolution GIS data in all three different types. We conducted two simulations using these two sets of GIS data. The ignition point of the low resolution simulation locates at (24, 24) and the ignition point of the high resolution simulation locates at (99, 99). Both simulations use the same wind speed and wind direction (from south to north). The simulation results on both resolutions at time = 7200 s are displayed on the right in Fig. 5a and b respectively. The simulation results show that the fire shape of low resolution simulation is similar to that of high resolution simulation at the given virtual time. This means the low resolution simulation is able to provide a good profile about the fire spread behavior (in a

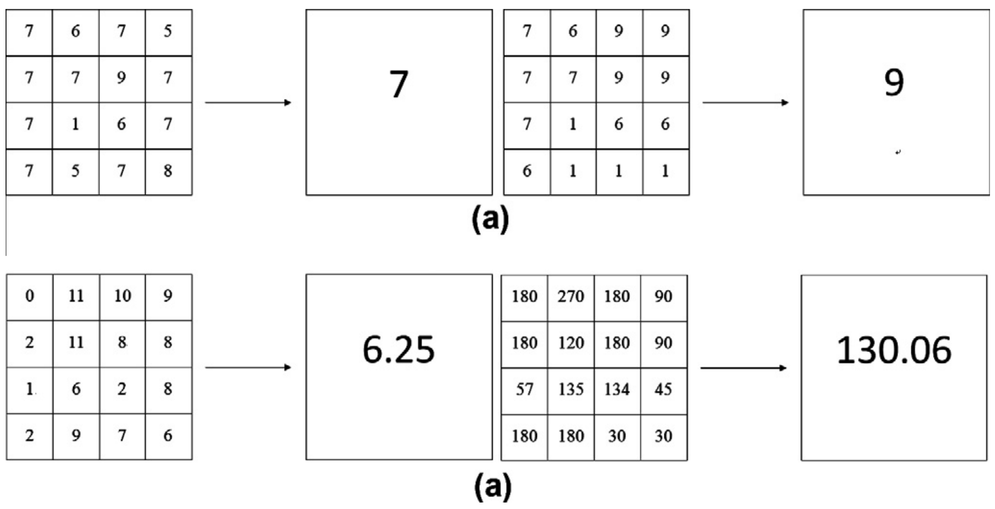


Fig. 4. Examples of “dominant fuel, averaged slope and aspect” rule: (a) “dominant fuel” and (b) “averaged aspect and slope”.

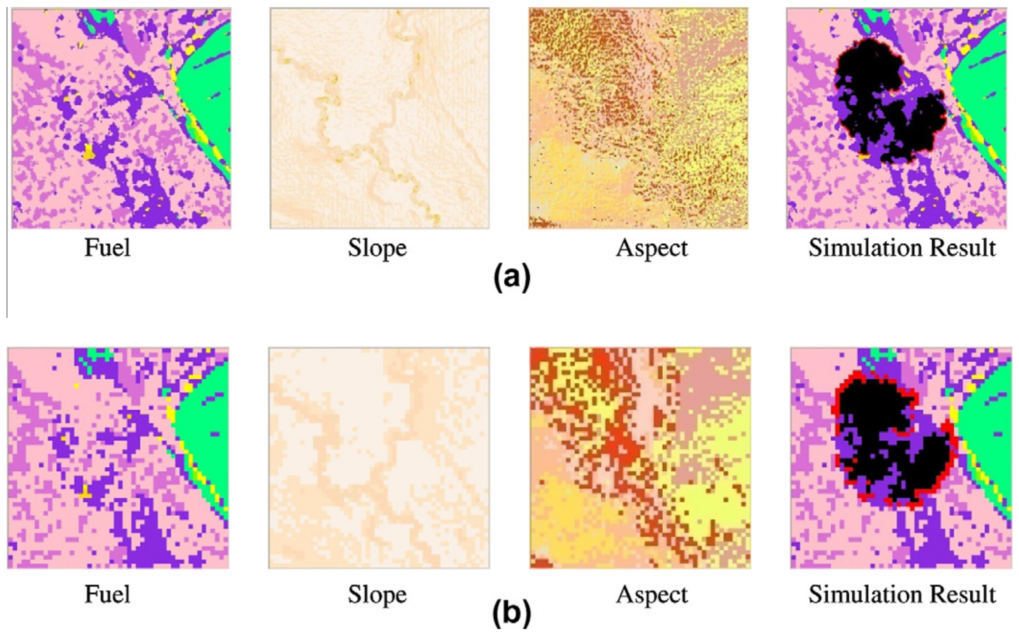


Fig. 5. An example using our “dominant fuel, averaged slope and aspect” algorithm: (a) high resolution GIS data and simulation result at time 7200 s (b) low resolution GIS data and simulation result at time 7200 s.

much faster execution time) of the high resolution simulation. However, it is important to note that the low resolution simulation cannot replace the high resolution simulation because it trades the simulation accuracy for speed.

4.2. Produce profile queue

The second step of the profile-based spatial partitioning method is to produce a profile using the generated low resolution GIS data. To produce the profile, we use the low resolution GIS data to run a low resolution simulation via DEVS-FIRE on a single machine. The low resolution simulation uses the same weather data as that in the high resolution simulation. The ignition point in the low resolution simulation is converted from that in the high resolution simulation based on the following formula, where x_{LR} and y_{LR} denote the x and y coordinates of the ignition point in the low resolution simulation, x_{HR} and y_{HR} denote the x and y coordinates of the ignition point in the high resolution simulation, and R is the resolution factor.

$$x_{LR} = x_{HR}/R; \quad y_{LR} = y_{HR}/R$$

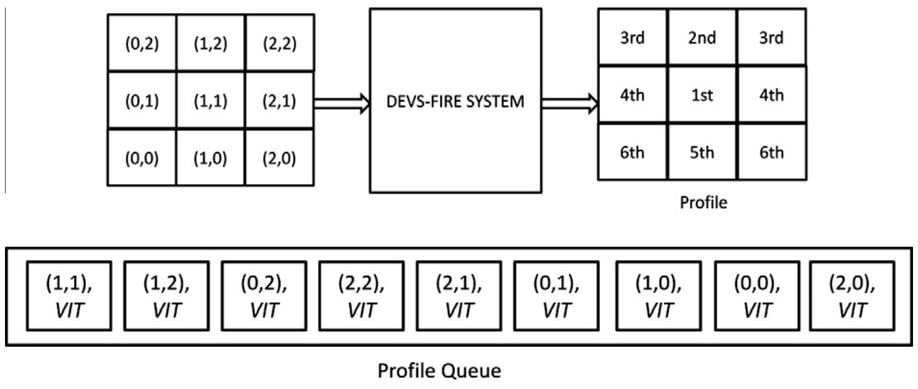


Fig. 6. Producing profile using low resolution GIS data: the sequence in the ***PQ*** represents the order this cell is ignited.

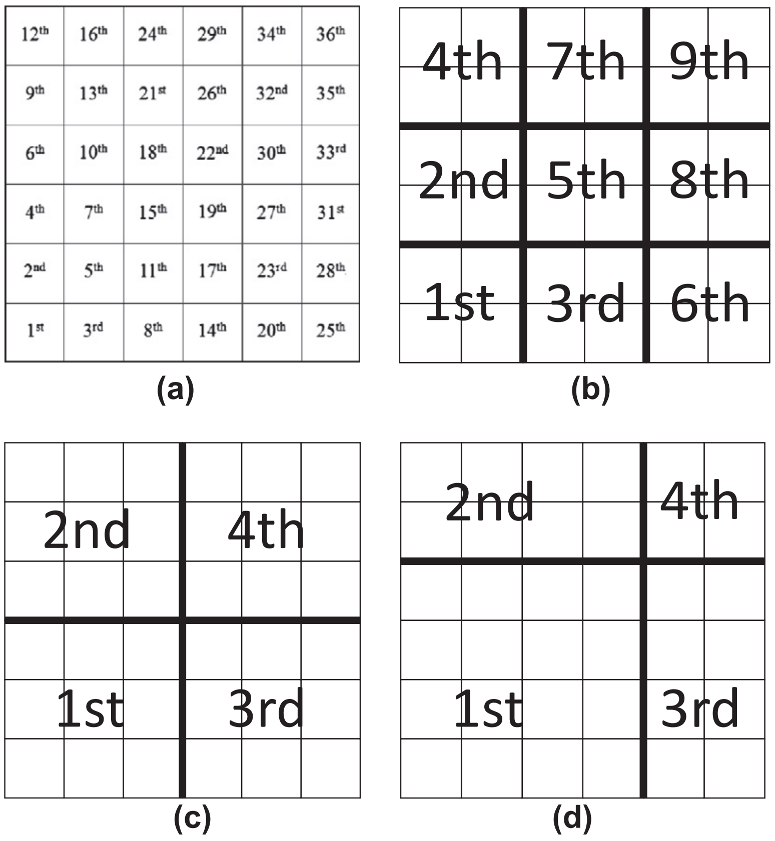

12th 16th 24th 29th 34th 36th
9th 13th 21st 26th 32nd 35th
6th 10th 18th 22nd 30th 33rd
4th 7th 15th 19th 27th 31st
2nd 5th 11th 17th 23rd 28th
1st 3rd 8th 14th 20th 25th
(a)
4th 7th 9th
2nd 5th 8th
1st 3rd 6th
(b)
2nd 4th
1st 3rd
(c)
2nd 4th
1st 3rd
(d)

Fig. 7. Examples of partitioning with different granularities: (a) $g = 1$, (b) $g = 2$, (c) $g = 3$, (d) $g = 4$.

In the low resolution simulation, once a cell is ignited, we record a virtual ignited time (*VIT*) that indicates when this cell is ignited. If a cell cannot be ignited, its *VIT* is infinity. After the simulation is finished, we sort the *VIT* of all cells from small to large and use the sequence of sorted cells to represent the fire spread behavior. In our work, a queue is used to store the profile. Each item in the queue contains the *x* and *y* coordinates of the cell as well as its corresponding *VIT*. We name this queue as the profile queue (denoted as ***PQ***). The profile queue includes two types of information: (1) the ignition sequence of the cells, and (2) the time of ignition of the cells. The profile queue is defined as follows,

$$\boldsymbol{PQ} = \{(x_{\mathrm{LR}}, y_{\mathrm{LR}}, VIT)_i | 0 \leqslant x_{\mathrm{LR}} < L_{\mathrm{LR}}, 0 \leqslant y_{\mathrm{LR}} < W_{\mathrm{LR}}, 0 \leqslant VIT_i \leqslant VIT_j, 0 \leqslant i < j < L_{\mathrm{LR}} W_{\mathrm{LR}}\}$$

where the subscript i denotes the order of the cell in the queue, x_{LR} and y_{LR} are the coordinates of the cell, L_{LR} is the length of the low resolution cell space and W_{LR} is the width of the low resolution cell space. Fig. 6 shows an example of how to produce the profile and ***PQ***. The top part of the figure shows the cells in the cell space (on the left) and their ignition sequence (on the right) in the low resolution simulation. The bottom part of the figure shows the profile queue based on the low resolution simulation. For simplicity, all cells' virtual ignition time are denoted as *VIT* in the figure.

4.3. Profile-based partitioning with spatial granularity

To support parallel simulation, the cellular space needs to be divided among multiple PUs. In the profile-based spatial partitioning, the cellular space is divided according to the profile queue (***PQ***) generated in the previous step. The basic idea is to allocate the items in ***PQ*** sequentially to the PUs. Note that each item in ***PQ*** represents a block of cells (specified by the resolution factor *R*) in the high resolution cell space. Allocating the items individually can effectively support load balance among the PUs. However, it may not achieve optimal result because each item is a relatively small block (especially when the resolution factor *R* is small). Dividing the space according to these small blocks will result in frequent message passing among the PUs and thus lead to high communication cost and cause more rollbacks in the time warp algorithm. To increase the block size, we introduce a spatial granularity factor (denoted as *g*) that allows the user to specify the granularity of partitioning.

With the granularity *g*, the partitioning method allocates a $g \times g$ sub cell space to a PU each time according to the profile queue. Fig. 7 illustrates examples of partitioning with different granularities. Fig. 7a shows the ignition sequence of the cells in a 6×6 cell space. If the granularity *g* equals to 1, the partitioning process of the cells strictly follows the profile: the first ignited cell is allocated to *PU1*, the second ignited cell is allocated to *PU*2, ..., the *n*th ignited cell will be allocated to *PUN* (where *N* is the total number of PUs), and the $(n + 1)$th ignited cell will be allocated to *PU1* again, and so on. Fig. 7b illustrates the partitioning process when the granularity *g* equals to 2. In this case, the ignition order of each 2×2 sub cell space is decided by the next earliest ignited cell it contains. For example, the 2×2 sub cell space with "1st" on it contains the first ignited cell; thus this sub cell space will be allocated to *PU1*. After that, since the second and the third ignited cells have already been assigned to *PU1*, the 4th ignited cell will be the next earliest ignited cell. Thus, the 2×2 sub cell containing this cell is marked as "2nd" and is allocated to *PU2*. This continues until the whole cell space is finished. Fig. 7c illustrates the partitioning sequence when the granularity *g* is 3 and Fig. 7d illustrates the partitioning sequence when the granularity *g* is 4.

We define the profile queue with granularity (denoted as ***PQG***) as the adjusted profile queue according to the granularity described above. Given a ***PQ*** and the granularity *g*, the algorithm of producing ***PQG*** is given below.

Algorithm (Producing PQG)

```
Input: profile queue PQ; granularity g; dimension of the low resolution cell space $W_{LR}$ and $L_{LR}$; number of
  participating PUs N.
Output: profile queue with granularity PQG.
  n ← 0;
  for each item in PQ, do
    if item is not allocated, do
      h_start ← (item. x/g) * g)
      h_end ← h_start + g
      if h_end > $L_{LR}$−1,do
        h_end ← $L_{LR}$−1
      end if
      v_start ←(item. y/g) * g
      v_end ← v_start + g
      if v_end > $W_{LR}$−1, do
        v_end ← $W_{LR}$−1
      end if
      for each item in PQ, do
        if h_start <= item.x < h_end and v_start <= item.y < v_end, do
          PQ.remove(item);
          PQG.push_back (item, n)
        end if
      end for
      n ← (n + 1)% N
    end if
  end for
  return PQG
```

After partitioning ***PQG***, each PU obtains a sub set of ***PQG***. Since each item in the ***PQG*** represents a block of cells in the high resolution cell space, the sub sets of ***PQG*** are then converted to the corresponding high resolution cells. These are the spatial partition for the PUs.

4.4. Parallel simulation based on the profile-based spatial partitioning

After the spatial partitioning, we run parallel simulation using the time warp algorithm for DEVS models [11]. Under a single PU environment, when an event that represents the fire spreading from one cell to another is scheduled by the simulator in DEVS-FIRE, an ignition message will be sent from the burning cell to the unburned cell. This is achieved by inserting the ignition message into the recipient's input message list in time stamp order. When the simulation is initiated under a parallel environment, an ignition message can be destined to a cell that is not located on the same PU. Thus, we divide the ignition messages into two categories: inter-messages and intra-messages. For a cell, intra-messages are sent from its neighbor cells that locate on the same PU. Since both the source and destination are on the same PU, intra-messages are directly inserted into the target cell's input messages list. Inter-messages are needed when the source cell and destination cell are not on the same PU. When an inter-message is initiated, it is sent out from the source cell to the destination cell and it is up to the destination cell to process this inter-message. When an inter-message is received, it will first be stored in a message queue (*MQ*) which holds all the inter-messages a PU receives. The *MQ* keeps all its messages in ascending order based on their timestamps (*TS*s). That is the message that is at the head of the *MQ* has the smallest *TS* and the message that is at the tail of the *MQ* has the largest *TS*.

Since we are using time warp algorithm, rollbacks will be produced. To reduce the rollbacks, we propose a mechanism called watch dog in this paper. The basic idea of the watch dog is to exploit the *VIT* in the profile as a "watch dog" to screen the incoming inter-messages (sent from other PUs). The goal is to use the *VIT* to distinguish "out-of-order" ignition messages, that is, the messages that will cause rollbacks, and postpone processing them. This is based on the assumption that the *VIT* obtained in the low resolution simulation holds some "truth" about the high resolution simulation and thus can be utilized. Below we describe how the watch dog works. Because a cell in the low resolution simulation represents a block of cells ($R \times R$ cells) in the high resolution simulation, in our method the *VIT* obtained in the low resolution simulation is shared by all the cells in the $R \times R$ block. This time is denoted as $T_{\text{watch\_dog}}$. The watch dog mechanism is checked only when receiving an inter-message sent from other PUs. Specifically, when a cell receives an inter-message, it checks if any cell within its $R \times R$ block is already ignited. If yes, the message is processed. This is because an ignited block has already processed inter-messages before and thus there is no need to hold follow-on inter-messages. However, if this block has not been ignited, the *TS* associated with the incoming message is compared with $T_{\text{watch\_dog}}$. If $TS > T_{\text{watch\_dog}}$, this means other inter-messages (from other PUs) that contain a timestamp smaller than *TS* are likely to be received later (because the low resolution simulation indicates the block should be ignited earlier). If we process this message to ignite the block, it may cause rollbacks when those messages with smaller *TS*s are received. Thus the watch dog mechanism holds this message without processing it. We note it is possible that the $T_{\text{watch\_dog}}$ obtained from the low resolution simulation is inaccurate. Thus to ensure the correctness of the parallel simulation, the held messages are not discarded and are stored in the *MQ* for later processing. As the simulation proceeds, when the local simulation time of the PU reaches a held message's ignition time, the held message will be re-processed. Therefore, even if the $T_{\text{watch\_dog}}$ obtained from the low resolution simulation is not precise, the watch dog mechanism will not cause errors in the parallel simulation. In the extreme case, all of the real ignition messages may have the *TS*s all greater than the $T_{\text{watch\_dog}}$ of the corresponding block, which may cause the simulation to be stalled. At this situation, we compare the smallest *TS* with the Global Virtual Time (*GVT*). If this *TS* is equal to the *GVT*, we will accept this message because it is time to process it. To update the *GVT*, we followed the algorithm in [21]. When an inter-message is sent or received, we update local Message Matrix (MM) and Table of Forcing Vectors (TFV) according to the steps in "Update on Send" or "Update on Receive" specified in [21]. If it is time to calculate *GVT*, the minimum of all the *lvt*s and the *t*s in all the forcings in the TFV is selected as the *GVT*. To focus on the watch dog, we omit the detail of *GVT* update part in the description of the algorithm. Next, we provide the algorithm of the simulation with the watch dog mechanism. In the algorithm, *WATCHDOG[lx][ly]* represents the virtual ignition time $T_{\text{watch\_dog}}$ obtained from the profile for partition block at (*lx*, *ly*), *message. TS* are the time stamps associated with the message. More information about the time warp algorithm for DEVS models can be found in [11].

Algorithm (watch dog mechanism)

```
Input: MQ; dimension of the low resolution cell space $W_{LR}$ and $L_{LR}$; resolution factor R;
while terminating condition is not met, do
// process messages in the message queue MQ. Note that the MQ contains not only new received messages,
  but also previously received ''out-of-order'' messages that might be screened out by the watchdog
  mechanism.
  for each message in MQ, do
    if message has not been used for GVT, update TFV and MM according to [21];
    lx ← message.x/R;
    ly ← message.y/R;
    if partition block at (lx, ly) is ignited or message.type is ROLLBACK, do
      remove message from MQ and insert message into the local target cell(message.x, message.y)'s
  input messages list;
```

```
    else if partition block at (lx, ly) is not ignited, do
      if message.TS == GVT or message.TS <= WATCHDOG[lx][ly] or message.TS <= LVT, do
        remove message from MQ and insert message into the local target cell(message.x,
  message.y)'s input messages list;
      else
        put messageback into the MQ;
        end if
      end if
    end for
    get all the imminents;
    for each imminent imm, do
        run the simulation using the time warp algorithm for DEVS models[11] on each imm. Specially,
  when sending out a message, it includes sending intra-messages to the cells on the same PU, inter-
  messages to the cells not on this PU. When sending messages to another PU, MM and TFV are updated and
  then piggy-backed in the inter-messages according to [21]. If this cell is ignited, the partition
  block that contains this cell is marked as ignited. When a collision between the external
  transition function and the internal transition function happens to the imm, the confluent
  function occurs, in which the internal transition function executes first and then the external
  transition function executes.
        // When imminents execute output functions, the output messages are transferred to be the
  input messages of some models. These models may be in imminent or may not. The models that are not in
  imminent are called influenced models.
        get all the influenced models;
        for each influenced model, do
          run the simulation using the time warp algorithm for DEVS models[11]on the influenced
  model.
      Reschedule the imminents and schedule the influenced models;
  update LVT by using the smallest lvt of all the scheduled models;
      update this PU's lvt in TFV using LVT
      if it is time to update GVT, do the calculation specified in [21];
  end while
```

In the above algorithm, we note that although multiple blocks may be partitioned together based on the granularity factor g, in the algorithm each block still uses its own $T_{\text{watch\_dog}}$. In other words, the granularity factor g acts as a partitioning factor only. It does not mean the different blocks in the same partition will use the same $T_{\text{watch\_dog}}$. We also note that the watch dog mechanism can only alleviate the problem of rollback to some extent. There are other (important) situations that the watch dog cannot prevent rollbacks. Performance analysis about how significant the watch dog can reduce rollbacks is given in Section 5.

5. Performance analysis

To help to see the performance impacts of the profile-based spatial partitioning method, in this section we present performance analysis for the profile-based spatial partitioning method. We carry out this analysis by comparing with a uniform spatial partitioning method where the cellular space is partitioned into size-equivalent sub cell spaces according to the number of PUs. The analysis includes both time complexity analysis and overhead analysis.

5.1. Time complexity analysis

Conceptually, the total execution time (denoted as T) of an optimistic parallel simulation consists of two parts: the time for computing the fire spread (denoted as T_c), and the overhead time (denoted as T_o) incurred by the optimistic parallel simulation algorithm. In wildfire spread simulation, it is important to note that a PU cannot start the simulation if no cell in its partition is ignited. In this case, the PU has to wait until an ignition message is received (from other PUs) to ignite its cells.

For the purpose of our analysis, T_c can be further broken into two parts: the time when only some (not all) PUs are active while other PUs are idle, and the time when all PUs are active in computing the simulation tasks. The first part refers to the situation where the computation loads among PUs are imbalanced, thus some PUs are active while others are idle. This time is denoted as $T_{\text{c\_imbalanced}}$. The second part refers to the situation where all PUs are active. This is denoted as $T_{\text{c\_balanced}}$. As a result,

$$T = T_{\text{c\_imbalanced}} + T_{\text{c\_balanced}} + T_o$$

In an ideal case where all PUs have the same processing speed, and the tasks are perfectly balanced (evenly distributed among the PUs), and the overhead time is zero,

$$T = T_{\text{fire\_spread}}/N = M^2 T_{\text{one\_cell}}/N$$

where M is the size of the cell space, N is the total number of participating PUs, $T_{\text{fire\_spread}}$ is the time spent in computing the entire fire spread simulation process, and $T_{\text{one\_cell}}$ is the time spent in simulating one cell, which can be treated as the same for different cells.

For a given fire spread simulation where the total computation tasks ($T_{\text{fire\_spread}}$) are fixed, to reduce T we should decrease $T_{\text{c\_imbalanced}}$ and increase $T_{\text{c\_balanced}}$ as much as possible and reduce T_o too. This means we should partition the cell space in a way that all PUs are active in computing the fire spread for some cells at most of the times during the simulation, regardless of the fire spread behavior. Next, we analyze the time complexity of two sample simulation scenarios for both the uniform spatial partitioning and the profile-based partitioning.

5.1.1. Time complexity analysis for the uniform spatial partitioning method

For our analysis we consider a uniform spatial partitioning where four PUs are used and the space is divided into four sub spaces as shown in Figs. 8 and 9. We analyze T_c for the uniform spatial partitioning method, which is influenced by the fire spread behavior. Since a pure theoretic analysis of time complexity is intractable due to the complexity of fire spread behavior, we consider two sample simulation scenarios with different fire spread behaviors as two study cases. In these examples, four PUs are employed and the size of the cell space is $M \times M$. Our analysis is based on the assumption that all the PUs have the same computing speed and the simulation ends when all the cells in the cell space are burned out. Note that in this section we do not consider the overhead (e.g., overhead due to rollback). The goal of this analysis is to compare with the profile-based partitioning method in terms of the parallelism.

Fig. 8 shows an example with one ignition point locating at the bottom left corner of the partition on *PU0* and the three figures represent three different simulation stages. In Fig. 8a, only *PU0* has been active so far. *PU1*, *PU2* and *PU3* are idle because the fire has not spread onto their partitions. After the stage shown in Fig. 8a and before the stage shown in Fig. 8b, *PU0*, *PU1* and *PU3* are active while *PU2* is still idle. After that the partition on *PU0* is finished and *PU1*, *PU2* and *PU3* are all active. This continues until to the stage as shown in Fig. 8c. Then *PU1* and *PU3* finish computing their partitions and only *PU2* is active. Therefore, in this example there is no time interval when all four PUs are active, which leads to poor parallelism.

Fig. 9 displays the second example, in which each partition has an ignition point as shown in Fig. 9a. In this case, since all four PUs have the same computing speed, the fire spreads as illustrated in Fig. 9. Thus, this case has a high level of parallelism because all four PUs run in parallel from the beginning to the end.

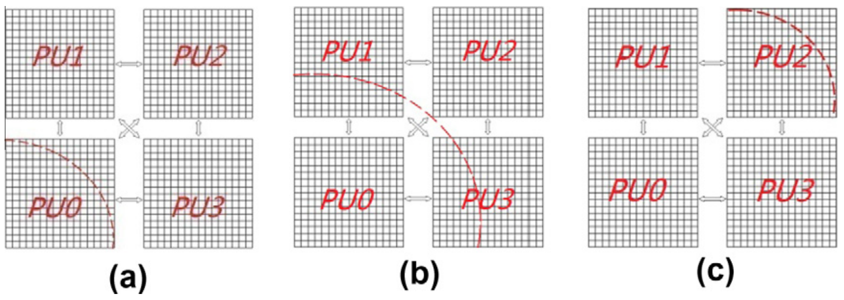


Fig. 8. One ignition point in the bottom left partition. The dashed line in (a–c) represents the front line of the fire at different simulation times.

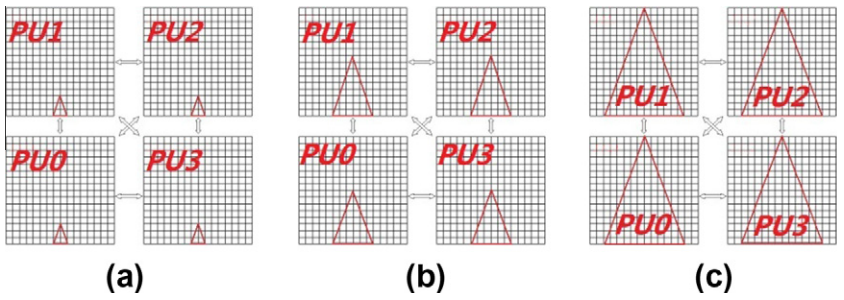


Fig. 9. Four ignition points and four partitions. The dashed lines in (a–c) represent fire spread at different simulation times.

5.1.2. Time complexity analysis for the profile-based spatial partitioning method

In this section, we analyze the time complexity for the profile-based spatial partitioning method using the same two fire spread examples describe above. Similarly, we consider four PUs and assume all PUs have the same processing speed. The resolution R is set to 4 and the granularity g is set to 2. For the profile-based spatial partitioning method, the time cost spent on producing the profile (denoted as $T_{c\_profile}$) can be represented by

$$T_{c\_profile} = (M/R)^2 T_{one\_cell}$$

This is the execution time for the low resolution simulation. When $R = 4$, $T_{c\_profile}$ is 1/16 of the execution time for the high resolution simulation on a single machine. In the experiments that will be described in Section 6, R is set to 10, thus the time for producing the profile is 1/100 of the high resolution time, which is insignificant.

Fig. 11 illustrates how the profile-based spatial partitioning method works for the two examples considered in the previous section. For clarity the figure displays only the low resolution cell space. The numbers in the figure represent the IDs of the PUs. From the figure we can see that in both cases all four PUs are running in parallel most of time. Comparing Figs. 10a and 8, one can see that at any stage there are more PUs engaged in the computation than that when using the uniform spatial partitioning method. The $T_{c\_imbalanced}$ in the profile-based partitioning method is much smaller because the space is divided into smaller blocks and assigned to the PUs according to the sequence of ignition. As the result, the fire in one PU can quickly spread to other PUs and thus engage them in the computation. Fig. 10b shows the partitioning result for the second

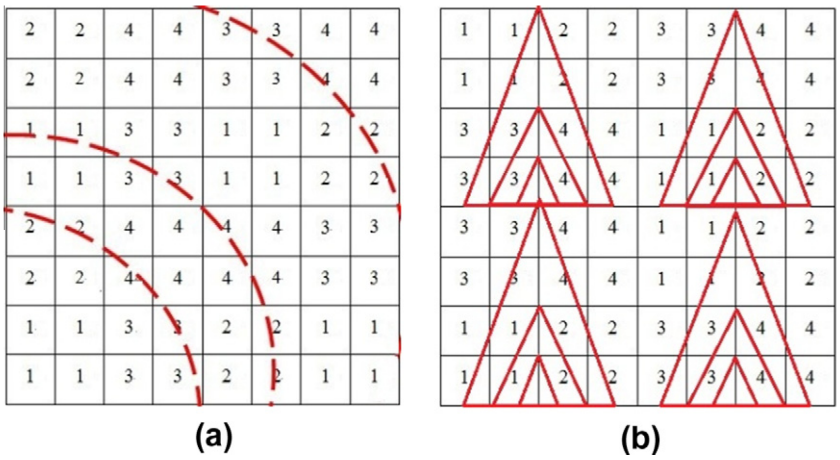


Fig. 10. Partitioning results using profile-based spatial partitioning $R = 4$ and $g = 2$.

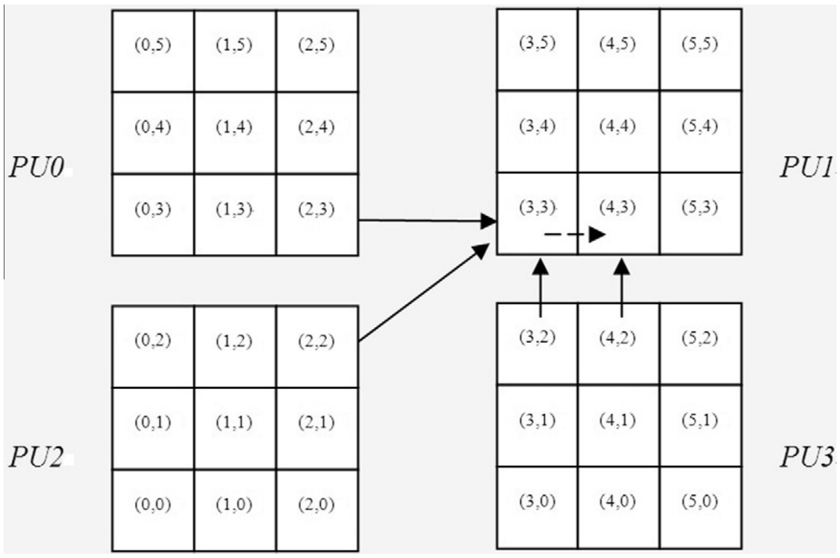


Fig. 11. A scenario after partition: $R = 3$.

example where there are four ignition points. Although the partitioning results are different from those in the uniform partitioning, the profile-based partitioning method results in the same high level of parallelism because all PUs start the simulation from the very beginning and last to the end. Fig. 10 illustrates an important feature of the profile-based partitioning method: the method can achieve high degree of parallelism regardless the fire spread behavior, e.g., where the ignition points are and which direction the fire spreads to.

From Fig. 10 one can also see how the granularity g can play a role in the simulation performance for the profile-based partitioning method. As g increases, the space is divided into larger blocks to be assigned to PUs. This has two aspects of impacts on the simulation performance. On one hand, the larger blocks may increase the portion of imbalanced computation because it takes longer for the fire in one PU to spread to other PUs. On the other hand, the larger blocks can reduce the communication overhead, and more importantly, can reduce the number of rollbacks because there is less message passing among the PUs. Some performance results measuring the impact of granularity g are provided in Section 6.2. We note the uniform spatial partitioning shown above can be considered as a special case of the profile-based partitioning by setting granularity g to be large enough so the space is divided into size-equivalent blocks according to the number of PUs.

5.1.3. Overhead analysis

The overhead analysis mainly focuses on the rollback of the parallel simulation. When using the uniform spatial partitioning, there are larger partitions and fewer partitioning boundaries. Thus, more communications among neighboring cells will be carried out by intra-messages. The profile-based spatial partitioning method results in smaller partitions and more partitioning boundaries. This means a lot of communications happen between neighboring cells via inter-messages instead of intra-messages. The increasing number of inter-messages potentially increases the number of rollbacks, and thus incurs more overhead than the uniform spatial partitioning method. Our experiment results in Section 6 show this.

As described in Section 4.4, a watch dog mechanism has been implemented to attempt to reduce the number of rollbacks. The goal of the watch dog mechanism is to use the $VIT_{\text{watch\_dog}}$ obtained from the low resolution simulation to filter the “out-of-order” messages and postpone processing them. Specifically, the watch dog mechanism can filter an “out-of-order” message (and thus reduce rollbacks) in the following condition:

$$VIT_{\text{watch\_dog}} < T_{\text{early\_but\_false}}$$

where $VIT_{\text{watch\_dog}}$ represents the virtual ignition time obtained from the low resolution simulation for the cell, and $T_{\text{early\_but\_false}}$ represents the time contained in the “out-of-order” message that arrives early and should be postponed processing. Fig. 11 illustrates one situation where the watch dog effects. In Fig. 11, suppose the cell $(3,3)$ of the block on PU1 should be ignited by the message from the cell $(3,2)$ of the block on PU3. Without the watch dog, the cell $(3,3)$ could be ignited by either the message from the cell $(2,3)$ of the block on PU2 or the message from the cell $(2,2)$ of the block on PU0 if those two messages arrives earlier than the message from cell $(3,2)$. If so, a rollback will be triggered when the right message from the cell $(3,2)$ arrives. However, if the watch dog is used and $VIT_{\text{watch\_dog}}$ can successfully screen out the incorrect messages from the cell $(2,3)$ or the cell $(2,2)$, the rollback will not be necessary. The watch dog mechanism can only alleviate the rollback problem to some extent due its dependency on the $VIT_{\text{watch\_dog}}$. If $VIT_{\text{watch\_dog}}$ is not accurate, the contribution of the watch dog mechanism is quite limited. We note that the effect of the watch dog mechanism is best observed when the PUs have different processing speeds. Section 6.4 shows some experiment results to demonstrate this.

To summarize, the profile-based partitioning method has the advantage of better parallelism and can be easily scaled to any number of PUs. On the negative side, it needs produce the profile by running a low resolution simulation and it incurs overhead due to more rollbacks. Although the uniform spatial partitioning method can work well in some special situations, its performance is greatly affected by the fire spread behavior.

6. Experiment results

In this section, several experiments based on different measurements are developed to test performance of the parallel simulation using both the uniform spatial partitioning method and the profile-based spatial partitioning method. Based on the Section 5, we consider several factors in our experiments: including number of PUs, granularity *g* and the initial locations of ignition points. We choose the values of these factors in order to demonstrate the advantages and limitations of the profile-based partitioning method. For example, we vary the ignition point location and the granularity factor g in different experiments to show how they affect the performance results. The experiment environment is on Cheetah, a Linux cluster with five nodes and four GPUs, 18.0G memory and CentOS release 5.4 (Final). The software employed in the experiments are mvapich2/gnu [36] and adevs-2.1 [35]. We implemented the time warp algorithm for DEVS models specified in [11] on top of adevs-2.1 by using MPI as our parallel mechanism. We also implemented our own version of [21] to update *GVT* on top of the time warp algorithm [11]. All the parallel simulations are conducted on a 1000×1000 cell space. In all of our experiments, the wind speed is constant and the wind direction is from south to north. The simulations stop when all the burnable cells in the 1000×1000 cell space are burned out. Each simulation is conducted five times, and the average of the experiment results is calculated. In all the experiments using the profile-based spatial partitioning method, the resolution factor is set to 10, which allows us to generate a reasonably meaningful profile in a fast time compared to the high resolution simulation. This means the low resolution simulation for producing the profile is based on a cell space with dimension 100×100. The

GVT is updated every ten seconds. The execution time results for the profile-based partitioning also includes the time spent on the low resolution simulation.

6.1. Simulations with different ignition locations and multiple ignitions points

This section shows how the fire spread behavior can influence the simulation performance for the two spatial partitioning methods. To produce different fire spread behavior we vary the location and the number of the ignition points in the wildfire spread simulations. For the profile-based spatial partitioning method, granularity *g* is set to 10. Together with the resolution factor of 10, this means each partitioning block for the spatial partitioning method corresponds to 100 $\times$ 100 cells in the high resolution simulation. Four PUs are used for both the uniform spatial partitioning method and the profile-based spatial partitioning method.

The first set of simulations (shown in Fig. 12) tests the impact of the ignition location on the simulation performance when there is only one ignition point. In this set of simulations, the locations of the only ignition points are set at (0, 0), (249, 249) and (499, 499) respectively. Fig. 12a illustrates the ignition points in the cell space under the uniform spatial partitioning method. When the ignition point locates at (0, 0), it is far away from the partitioning boundary. When the ignition point locates at (499, 499), it is right at the partitioning boundary. When the ignition point locates at (249, 249), it is at the center of the partition on *PU0*. Fig. 12b shows the ignition point in the cell space under the profile-based spatial partitioning method. Compared to the uniform spatial partitioning, there are more partitions and each partition is much smaller. When the ignition point locates at (0, 0), it is much closer to the partitioning boundary in Fig. 12b than that in Fig. 12a. When the ignition point locates at (249, 249) or (499, 499), they are right at the partitioning boundary.

Fig. 13 shows the execution time (Fig. 13a) and the number of rollbacks (Fig. 13b) for both the profile-based spatial partitioning method and the uniform spatial partitioning method. For the profile-based spatial partitioning, the time spent on producing the profile and the partitioning process is also included in Fig. 13a, which is only a small portion of the execution time as shown in the figure. To show the effect of the parallel simulations Fig. 13a also displays the simulation execution time when using a single-processor (denoted as 1PU in Fig. 13a). From the figure, we can see that when using the uniform spatial partitioning the execution time decreases as the ignition point moves towards the center of the space. This is expected because the center of the space is right to the partitioning boundary. The closer the ignition point locates to the

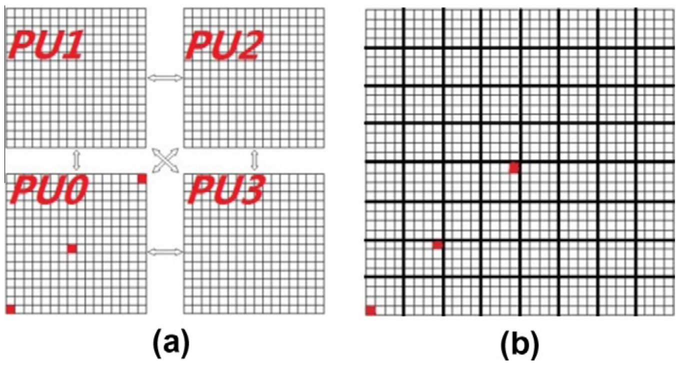


Fig. 12. Different locations of the ignition point in the cell space: (a) uniform partitioning (b) profile-based partitioning.

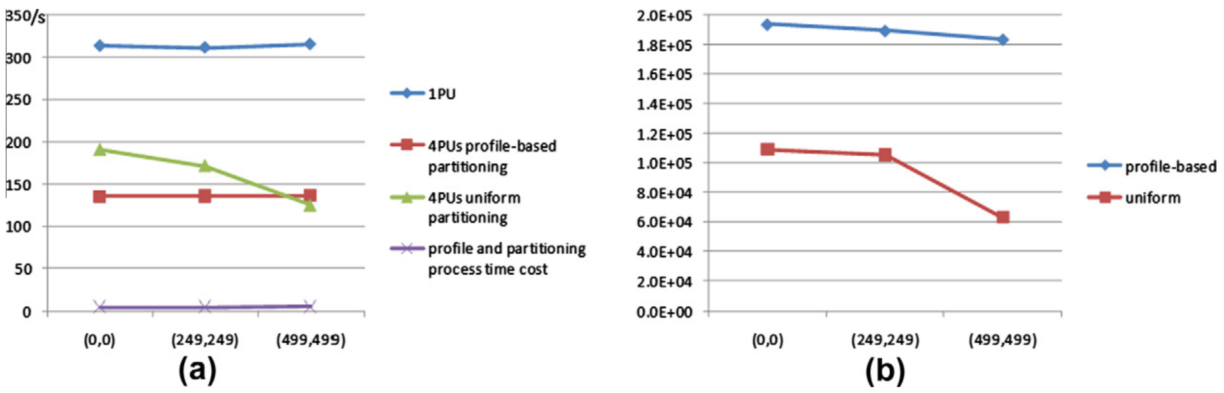


Fig. 13. Experiment result: (a) execution time (b) total number of rollbacks of all PUs.

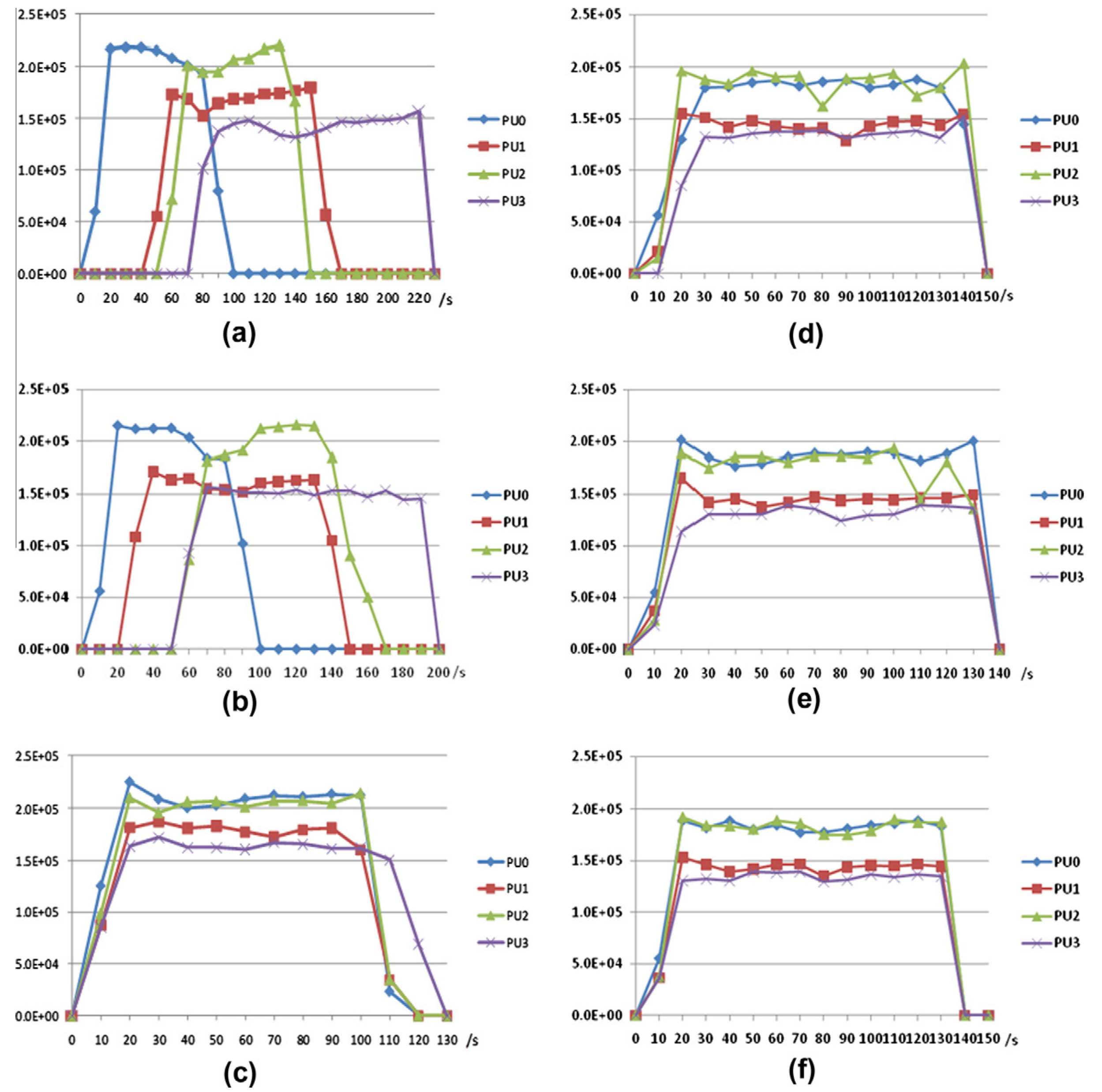


Fig. 14. Work load diagrams: (a) uniform partitioning and ignition point at (0, 0), (b) uniform partitioning and ignition point at (249, 249), (c) uniform partitioning and ignition point at (499, 499), (d) profile-based partitioning and ignition point at (0, 0), (e) profile-based partitioning and ignition point at (249, 249), (f) profile-based partitioning and ignition point at (499, 499).

partitioning boundary, the faster the fire spreads onto other partitions on other PUs. Different from that in the uniform spatial partitioning, the execution time of the profile-based partitioning is more stable when the ignition location changes. The execution time for location (0, 0) is a little bit higher than those for the other two locations. This is because the other locations are right at the partitioning boundary thus it takes less time for the fire to spreads to all four PUs. Fig. 13a also shows when the ignition point locates at (0, 0) and (249, 249), the profile-based spatial partitioning has better performance (less execution time) than the uniform spatial partitioning. This is because both these two locations are far away from the partitioning boundary (see Fig. 12a) for the uniform spatial partitioning. Thus the computation load is unbalanced in the first stage of the simulation for the uniform spatial partitioning because the fire exists in only one PU. When the ignition point is at (499, 499), the uniform spatial partitioning produces better performance result. This is mainly due to the number of rollbacks as explained below.

Fig. 13b shows the total number of rollbacks of all PUs for the two partitioning methods. The result confirms our analysis in Section 5 that the profile-based spatial partitioning produces more rollbacks than the uniform spatial partitioning does.

Table 1
The speed up of both partitioning methods.

Ignition locations	(0, 0)	(249, 249)	(499, 499)
Profile-based spatial partitioning	2.41	2.48	2.46
Uniform spatial partitioning	1.65	1.82	2.68

Table 2
Locations of the ignitions.

Name	Ignition locations
Random 4(a)	(388, 595), (148, 102), (993, 870), (997, 463)
Random 4(b)	(18, 26), (487, 17), (19, 387), (492, 267)
Random 20	(419, 30), (431, 114), (127, 125), (798, 860), (133, 174), (311, 513), (979, 592), (155, 701), (691, 810), (779, 277), (531, 289), (186, 879), (342, 210), (946, 521), (207, 853), (261, 626), (236, 44), (92, 715), (521, 890), (576, 654)

When the ignition point locates at (0, 0) and (249, 249), the profile-based method produces about 80,000 more rollbacks in total than that of the uniform partitioning method. When the ignition point locates at (499, 499), the profile-based partitioning method produces about 120,000 more rollbacks than that of the uniform partitioning method. In this case, both partitioning methods lead to balanced work load among the PUs (see Fig. 14 later). However, the less number of rollbacks make the uniform spatial partitioning has better performance. Table 1 lists speedup of both partitioning methods compared to the single-processor simulation for the different ignition locations. The speedup of the profile-based partitioning is stable for different ignition locations, while the speedup of the uniform spatial partitioning varies as the ignition location changes.

To better examine the execution time and its relationship with the load balance, Fig. 14 shows the work load of the two methods. The horizontal axis represents the execution time and the vertical axis represents amount of computation (calculated as the number of simulation iterations) during every time interval of 10 s. The figure confirms that for the profile-based partitioning, all four PUs participated in the computation from the very beginning and lasted to the end of the simulation. However, for the uniform spatial partitioning, the PUs have balanced work load only for the case of location (499, 499). In the two other cases, different PUs started and ended their computations at different time points. For example, in Fig. 14a, in the first 40 s only PU0 was active and then from time 170 s, only PU3 was active while all other PUs were idle. The profile-based partitioning method is superior to the uniform spatial partitioning method in temporally balanced workload as shown in Fig. 14, and thus leads to better performance results.

To further examine how fire spread behavior can influence the simulation performance, our second set of simulations use multiple ignition points. We keep all the other configurations same as before, except that the number of the ignition points is increased. The locations of the ignition points in this set of simulations are randomly generated. Table 2 shows the number and locations of the ignition points. In random 4(a) and random 4(b), four ignition points with random locations are used. In random 20, twenty ignition points with random locations are used. If the uniform spatial partitioning method is used, there is one ignition point on each PU for the situation random 4(a). For the situation random 4(b), the four ignition points locate on one PU. For the situation random 20, there is at least on ignition point on each PU. If the profile-based partitioning method is used, each PU will obtain at least one ignition point in all three situations in Table 2. Fig. 15 shows the experiment results of all three situations using both partitioning methods.

From Fig. 15, one can see that the uniform spatial partitioning method has slightly better performance when there is at least one ignition point on each PU as shown in random 4(a) and random 20. This is because in these cases all PUs can start the simulation at the very beginning and achieve high parallelism. For the case random 4(b) where all four ignition points

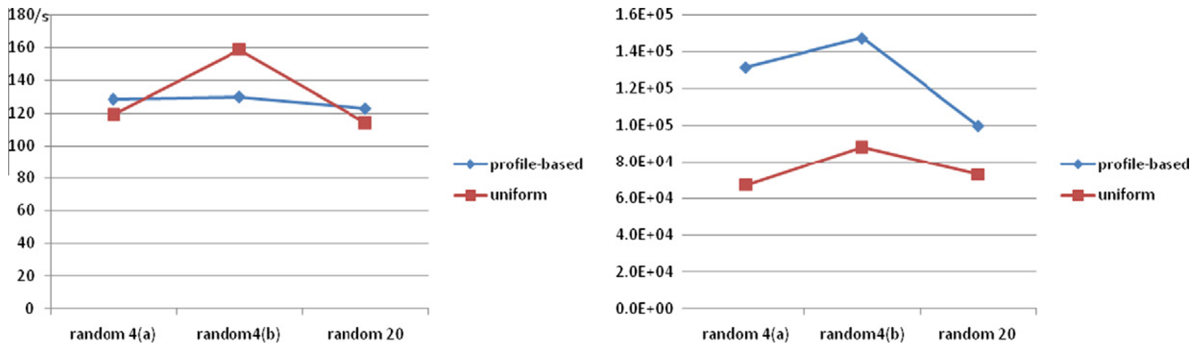
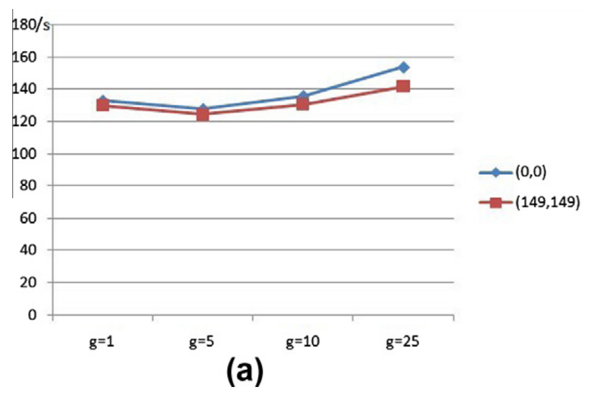


Fig. 15. Experiment results: (a) execution time (b) total number of rollbacks of all PUs.



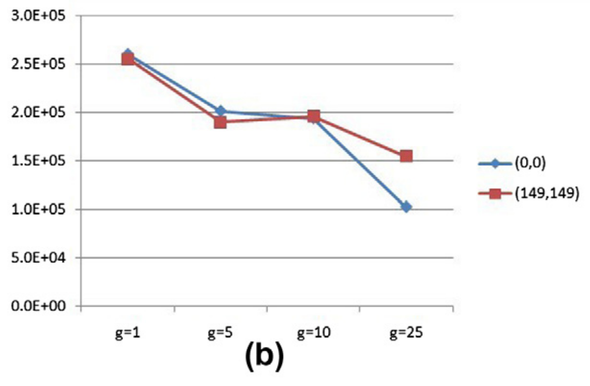


Fig. 16. Experiment results: (a) execution time (b) total number of rollbacks of all PUs.

locate in one partition, the execution time of the uniform spatial partitioning increases and is larger than that of the profile-based partitioning. This is due to the poor parallelism as explained before. On the contrary, the performance of the profile-based partitioning is stable as the ignition points vary. This set of experiments confirms with the previous experiments that the profile-based partitioning was able to achieve good parallelism independent of the fire spread behavior.

6.2. Simulations with different granularities

This experiment varies the granularity g and shows the impact of this factor on the performance of the profile-based spatial partitioning method. Similar as before, we use four PUs to run the parallel simulations. The granularity g is set to 1, 5, 10, and 25 respectively. We use only one ignition point and its location is set at (0, 0) in the first case and (149, 149) in the second case. Fig. 16 shows the execution time and the number of rollbacks as the granularity g increases.

For the first case where the ignition point locates at (0, 0), Fig. 16a shows that the execution time is reduced when g increases from 1 to 5. The total number of rollbacks is also reduced as g increases from 1 to 5. When $g = 1$, each partition block is a 10×10 ($R * g = 10$) sub cell space. When $g = 5$, each partition block is a 50×50 ($R * g = 50$) sub cell space. Since the fire is ignited at (0, 0), it takes more time for the fire to spread out of a 50×50 cell space (thus to ignite other partitions in other PUs) than that of a 10×10 cell space. However, as analyzed the number of rollbacks will be reduced when the granularity g increases. Fig. 16b shows that the number of rollbacks decreases about 50,000 when g increases from 1 to 5. Thus, even though it takes more time for the fire to spreads out of a 50×50 cell space, the decrement of rollbacks dominates the time difference and makes the execution time reduced when g increases from 1 to 5. When g further increases to 10, the size of each partition block increases to 100×100 ($R * g = 100$) while the number of rollbacks only slightly drops a little. In this case, the time spent on spreading out of the 100×100 sub cell space dominates. Therefore, the execution time increases as g increases from 5 to 10. When g is 25, the size of each partition block is 250×250 ($R * g = 250$). The number of rollbacks drops about 100,000. However, it takes longer time for the fire to spread out of the partition block and this time dominates. As a result, the execution time increases too.

For the second case where the ignition point locates at (149, 149), the result curves are similar to those in the first case but with less execution time. This is because when g is 1 and 5, ignition point (149, 149) locates at the partitioning boundary. The fire will immediately spread onto other partitions on other PUs. When g is 10 and 25, the ignition point (149, 149) locates much closer to the partitioning boundary than in the first case. Thus, it takes less time for the fire originating from the ignition point (149, 149) to reach the partitioning boundary, which leads to less execution time as shown in Fig. 16a.

This experiment shows that the granularity g can have impact on the simulation performance. Different g leads to different partitioning results. In general, as g increases, the size of the partition block increases. This means it takes longer time for a fire to spread to other partition blocks. Thus it increases the simulation execution time due to unbalanced work load, e.g., one PU is active while others are waiting for the fire to spread to their partitions. On the other hand, when g increases, the number of rollbacks reduces because the number of message passing among PUs is reduced. The less number of rollbacks decreases the performance overhead (see analysis in Section 5). Therefore, how to choose the granularity g needs to consider both sides of the performance impacts as explained above.

6.3. Simulations with different number of pus

This experiment varies the number of participating PUs to test the scalability of the profile-based spatial partitioning method. In these simulations, the resolution R is 10 and granularity g is 10. Thus, the 1000×1000 cell space is divided into one hundred 100×100 blocks. The ignition point is set to (0, 0). The number of participating PUs varies from 4 to 8. As a result, each PU will obtain the same amount of cells when the number of participating PUs is 4 and 5. When the number

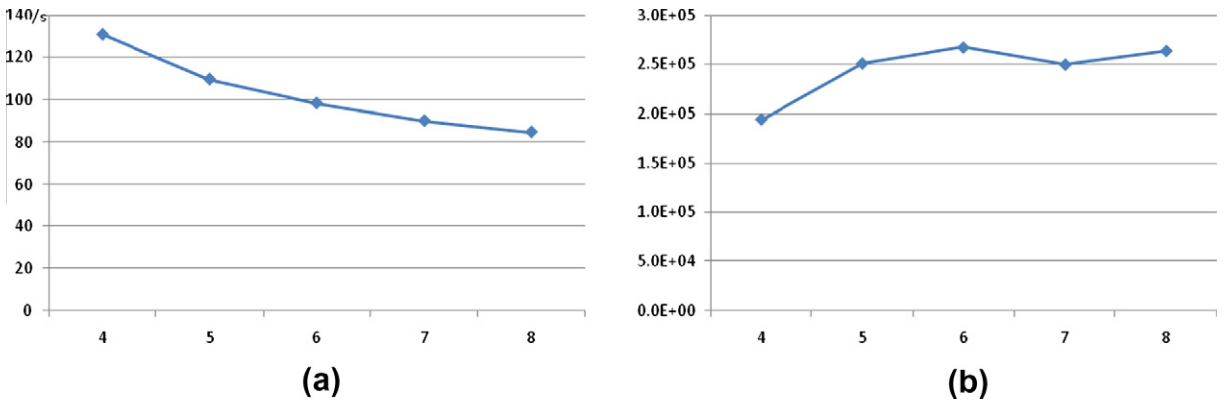


Fig. 17. Experiment results: (a) execution time (b) total number of rollbacks of all PUs.

Table 3
Speedup of different number of PUs.

	4	5	6	7	8
Speedup	2.41	2.86	3.19	3.49	3.70

of participating PUs is 6, 7 and 8, the load is a little unbalanced on each PU. The load difference is 10,000 (one 100×100 block) cells. In this experiment, we only consider the profile based method. This is because the uniform spatial partitioning method works best when the number of PUs is k^2 (k is an integer) [23] or there are ignitions in each partition after the uniformly spatial partitioning (see Section 5). When there is only one ignition point, the execution is nearly linear: PU containing the ignition point will run first and all other PUs will wait until an ignition message is received. Fig. 17 show the experiment results, from which we can see that the execution time decreases as the number of participating PUs increases. Table 3 lists the speedup of the parallel simulations compared to a single-processor simulation.

Although the execution time decreases as the number of PUs increases, the curve of decreasing becomes less steep as shown in Fig. 17a. This is partially due to the nature of wildfire spread behavior and how the profile-based partitioning method works. Based on the profile-based spatial partitioning method, some PUs will obtain their first partitions that are close to the ignition points; while other PUs will obtain their first partitions having a distance from the ignition point. As the number of PU increases, the distance between the ignition point and the first partitions of some PUs will increases as well. Together with the nature of fire spread behavior (a partition has no computation until it is ignited); some PUs will have to wait until the fire ignites their first partitions to start their simulations. The larger the number of PUs, the more PUs will wait and the longer time they will wait. To the extreme case when the number of PUs is equal to the number of partitions, the execution of the simulation is nearly sequential. This is no better than running on a single machine.

6.4. Simulations with “Watch Dog” mechanism

In this section we examine the “watch dog” mechanism described in Section 4.4 for reducing rollbacks. In this experiment, we employ two ignition points as shown in Fig. 18: one at (249, 249) and the other at (999, 999). Four PUs are used. For the profile-based spatial partition, the resolution R is set to 10 and granularity g is set to 5. In our experiment, *LIMIT* is set

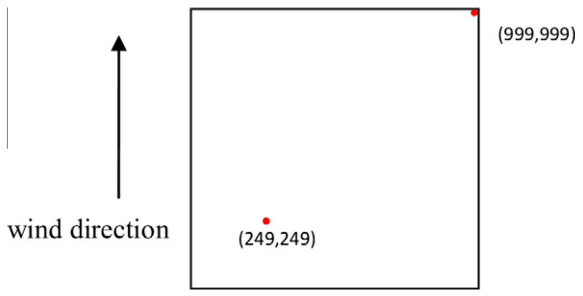


Fig. 18. Experiment scenario.

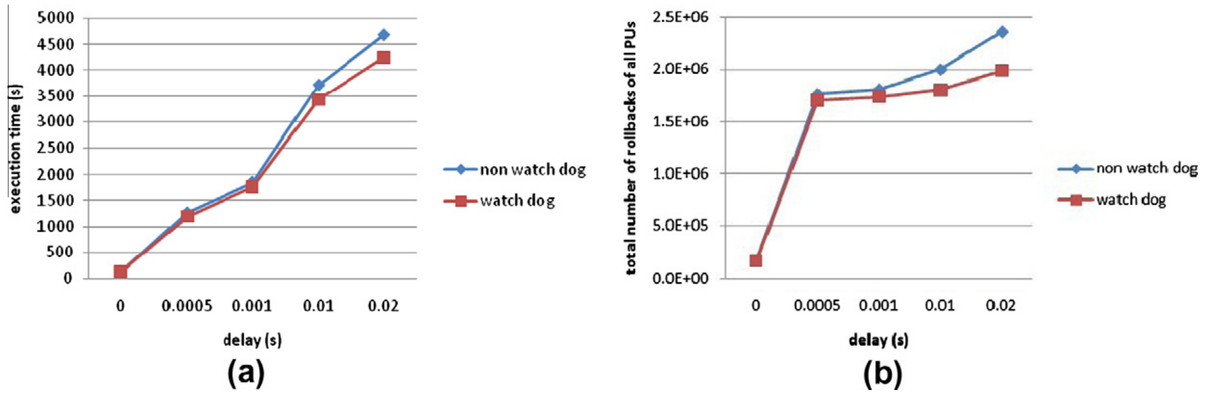


Fig. 19. Experiment results: (a) execution time (b) total number of rollbacks of all PUs.

to 20. We carry out the experiments using simulations with the watch dog mechanism and simulations without the watch dog mechanism and compare their experiment results.

Since the wind direction is from south to north, the burning area originating from ignition point (249, 249) should be larger than that originating from ignition point (999, 999) (because the fire is spreading to north). However, in a parallel simulation environment, the burning area originating from ignition point (999, 999) can be larger than the burning area originating from ignition point (249, 249) if ignition point (999, 999) is assigned to a faster PU and ignition point (249, 249) is assigned to a slower PU. If that is the case, when the burning area originating from ignition point (249, 249) eventually encounters the burning area originating from ignition point (999, 999), rollbacks will be triggered to ensure the correctness of the parallel simulation. The more area mistakenly "burnt" originating from ignition point (999, 999) which otherwise should be burnt originating from ignition point (249, 249), the more rollbacks will be triggered later on when the two burning areas encounter. When the watch dog mechanism is implemented, each partition block can check the ignition time of the incoming ignition message and compare it with the watch dog time $VIT_{\text{watch\_dog}}$ obtained from the low resolution simulation to screen out the potential "out of order" incoming ignition messages. To better demonstrate this effect of the watch dog mechanism, we make the PU containing ignition point (999, 999) faster than other PUs by adding a delay to all other PUs. The delay is achieved by the sleep system call and it is varied from 0 s (which means no delay) to 0.02 s. We measure the performance difference between the one using watch dog and the one without watch dog to see the effects of watch dog when there is difference in computing speed among participating PUs. Fig. 19 shows the experiment results.

From Fig. 19 we can see that the simulation performance is improved by using the watch dog mechanism. When the delay is zero, the watch dog nearly has little effect. This is because all the PUs have the same processing speed so that the fire originating from ignition point (999, 999) does not burn too much area. As the delay increases, more area is burnt by the fire originating from ignition point (999, 999). Thus, the number of rollbacks increases as well. After the watch dog is used, both the number of rollbacks and the execution time are reduced. The gaps between the two curves in both figures become larger when the delay increases. This indicates the watch dog screens out more "out of order" ignition messages. From this experiment we can see that the effect of the watch dog becomes more obvious when the PUs' processing speeds vary largely.

7. Conclusions

We present a partitioning method named profile-based spatial partitioning for parallel simulation of large scale wildfires. The profile-based partitioning exploits the dynamic behavior of a wildfire spread model and uses it as a profile to guide the spatial partitioning for parallel simulations. The profile is generated from a low resolution simulation using low resolution GIS data. It contains information including the ignition sequence and the time of ignition of the cells, which is exploited by the partitioning method. Performance analysis including the advantage and overhead of the profile-based partitioning method is carried out. Experimental results show that the profile-based partitioning leads to stable performance improvement regardless of the wildfire spread behavior. It increases the degree of parallelism and improves the scalability of parallel simulations of large-scale wildfires. On the negative side, it incurs more rollbacks due to the smaller size of the partition blocks. Future work includes developing methods to further exploit the profile of fire spread to develop dynamic partitioning methods based on the profile for improving the performance of large-scale wildfire simulations, and developing better ways to make watch dog more accurate. We will also study if the profile-based partitioning method can be generalized to other spatial–temporal simulations similar to the simulations of wildfire spread.

Acknowledgments

This research was supported in part by Grants CNS-0841170 and CNS-0941432 from the National Science Foundation.

References

[1] F. Kuhl, R. Weatherly, J. Dahmann, Creating Computer Simulation Systems: An Introduction to the High Level Architecture, Prentice Hall PTR, 1999. p. 1.
[2] B.P. Zeigler, T.G. Kim, H. Praehofer, Theory of Modeling and Simulation, second ed., Academic Press, New York, USA, 2000.
[3] A. Boukerche, Load balancing for parallel PCS networks simulation, Journal of Interconnection Networks 1 (3) (2000) 173–193.
[4] D. Chen, R. Ewald, G.K. Theodoropoulus, R. Minson, T. Oguara, M. Lees, B. Logan, A.M. Uhrmacher, Data access in distributed simulation of multi-agent systems, Journal of Systems and Software 81 (12) (2008) 2345–2360.
[5] F. Cicirelli, A. Furfaro, L. Nigro, An agent infrastructure over HLA for distributed simulation of reconfigurable systems and its application to UAV coordination, Simulation – Transactions of the Society for Modeling and Simulation International 85/1 (2009) 17–32.
[6] K.M. Chandy, J. Misra, Asynchronous distributed simulation via a sequence of parallel computations, Communications of the ACM 24 (11) (1981) 198–205.
[7] R.M. Fujimoto, Parallel discrete event simulation, Communications of the ACM 33 (10) (1990) 30–53.
[8] J. Hershberger, N. Shrivastava, S. Suri, C.D. Ťoth, Adaptive spatial partitioning for multidimensional data streams, Algorithmica 46 (1) (2006) 97–117.
[9] X. Hu, Y. Sun, L. Ntaimo, Design and application of formal discrete event wildfire spread and suppression models, Simulation (2011), accepted for publication.
[10] D.R. Jefferson, Virtual time, ACM Transactions on Programming Languages and Systems 7 (3) (1985) 404–425.
[11] J. Nutaro, On constructing optimistic simulation algorithms for the discrete event system specification, Transactions on Modeling and Computer Simulation 19 (1) (2008) 21.
[12] L. Ntaimo, X. Hu, Y. Sun, DEVS-FIRE: towards an integrated simulation environment for surface wildfire spread and containment, Simulation 84 (4) (2008) 137–155.
[13] D. Sislak, P. Volf, M. Jakob, M. Pechoucek, Distributed platform for large-scale agent-based simulations, Lecture Notes in Computer Science 5920 (2009) 16–32.
[14] P. Andrews, C. Bevins, R. Seli, Behaveplus Fire Modeling System, Version 3.0: User's Guide, General Technical Report RMRS-GTR-106WWW Revised US Dept. of Agriculture, Forest Service, Rocky Mountain Research Station, Ogden, Utah, 2005.
[15] M. Finney, FARSITE: Fire Area Simulator-model Development and Evaluation, Research Paper RMRS-RP-4, US Dept. of Agriculture, Forest Service, Rocky Mountain Research Station, Ogden, Utah, 1998.
[16] R. Rothermel, A Mathematical Model for Predicting Fire Spread in Wildland Fuels, Research Paper INT-115. Ogden, UT: US Department of Agriculture, Forest Service, Intermountain Forest and Range Experiment Station, 1972, 40 p.
[17] A.H. Buss, D.K. Ahner, Dynamic allocation of fires and sensors (DAFS): a low-resolution simulation for rapid modeling, in: Proc. of the 2006 Winter Simulation Conference, 2006, pp. 1357–1364.
[18] A.C.H. Chow, B.P. Zeigler, Parallel DEVS: a parallel, hierarchical, modular modeling formalism and its distributed simulator, in: Proc. of the 1994 Winter Simulation Conference, 1994, pp. 716–722.
[19] C.G. Cassandras, C.G. Panayiotou, G. Diehl, W.-B. Gong, Z. Liu, C. Zou, Clustering methods for multi-resolution simulation modeling, in: Proc. of SPIE's 14th Annual International Symposium on Aerospace/Defense Sensing, Simulation, and Control, 2000, pp. 37–48.
[20] D.T. Drewry, W.R. Emanuel, An optimization-based multi-resolution simulation methodology, in: Proc. of the 2002 Winter Simulation Conference, 2002, pp. 467–475.
[21] E. Deelman, B.K. Szymanski, Continuously monitored global virtual Time, in: International Conference Parallel and Distributed Processing Techniques and Application, 1997, pp. 1–10.
[22] S. Guo, X. Hu, Y. Sun, Exploring Spatial partition for parallel simulation of DEVS-FIRE, in: Proc. Of Summer Computer Simulation Conference (SCSC'09), 2009.
[23] S. Guo, X. Hu, Profile-based partition for parallel simulation of DEVS-FIRE, in: Proc. of 43rd Annual Simulation Symposium (ANSS), 2010, pp. 155–155.
[24] J. Himmelspach, R. Ewald, S. Leye, A.M. Uhrmacher, Parallel and distributed simulation of parallel DEVS models, in: Proc. of the 2007 Spring Simulation Multiconference, 2007, pp. 249–256.
[25] X. Hu, L. Ntaimo, Dynamic multi-resolution cellular space modeling for forest fire simulation, in: Proc. of DEVS Integrative M&S Symposium (DEVS'06), Spring Simulation Multiconference, 2006.
[26] D. Jefferson, H. Sowizral, Fast concurrent simulation using the time warp mechanism, in: Proc. of the SCS Distributed Simulation Conference, SCS Simulation Series, 1985, pp. 63–69.
[27] B. Logan, G. Theodoropoulos, The distributed simulation of multi-agent systems, Proceedings of the IEEE 89 (2) (2001) 174–185.
[28] Q. Liu, G.A, Wainer, Accelerating large-scale DEVS-based simulation on the cell processor, in: Proc. of 2010 Spring Simulation Conference (SpringSim10), DEVS Symposium, 2010, pp. 124–124.
[29] B. Nandy, W.M. Loucks, On a parallel partitioning techniques for use with conservative parallel simulation, in: Proc. of the 7th Workshop on Parallel and Distributed Simulation, 1993, pp. 43–51.
[30] A. Park, R.M. Fujimoto, A scalable framework for parallel discrete event simulations on desktop grids, in: Proc. of the 8th IEEE/ACM International Conference on Grid Computing, 2007, pp. 185–192.
[31] V. Suryanarayanan, R. Minson, G.K. Theodoropoulus, Synchronized range queries, in: Proc. of DSRT (Distributed Simulation and Real-Time Application), 2009, pp. 41–47.
[32] Y. Sun, X. Hu, Partial-modular DEVS for improving performance of cellular space wildfire spread simulation, in: Proc. 2008 Winter Simulation Conference (WSC'08), 2008, pp. 1038–1046.
[33] Y. Zhang, G. Li, K. Huang, Optimizing model interoperability in parallel discrete event simulation for cluster environment, in: Proc. of the 2007 Summer Computer Simulation Conference, 2007, pp. 698–705.
[34] M. Morais, Comparing Spatially Explicit Models of Fire Spread through Chaparral Fuels: A New Model based Upon the Rothermel Fire Spread Equation, Master's Thesis, The University of California, Santa Barbara, 2001.
[35] adevs, <http://www.ornl.gov/~1qn/adevs/index.html>.
[36] mvapich2/gnu, <http://mvapich.cse.ohio-state.edu/>.