ForeFire open source wildfire front propagation model solver and programming interface

Jean-Baptiste Filippi
Corte, 20250 France
filippi@univ-corse.fr

CNRS, University of Corsica, UMR SPE 6134

Abstract

ForeFire code is an API, a library and an interpreter with a "pythonesque" syntax available under the GPL open source licence. Its purpose is to simulate the evolution of reactive fronts on a large domain at high resolution, ad is particularly applicable to rapid (sub-minute) simulation of large wildland fire. It is designed such as it can easily be run from the command line, coupled with other codes and extended with user defined propagation and surface emission (fluxes) models. Users of the code can therefore easily extend it to use it as a numerical testbed for new models. The code includes a solver that tracks the propagation of a fire front line. The fire front moves upon the earth surface according to several factors such as topography, fuel and wind. An analytical model is used to represent these effects on the propagation velocity of the front. The front is then tracked by approximating the fire front line with connected Lagrangian markers called nodes. Advection of these nodes according to the propagation velocity is carried through a constant-CFL asynchronous methodology that focus on zones of high velocity.

1 Introduction

ForeFire code has been developed to bridge the gap between research and operational code, based on [Filippi *et al.(2009)*], [Filippi *et al.(2011)*] and [Balbi *et al.(2009)*]. It is now open source, designed for large scale fire simulation, can be easily extended with any new model formulations and can take typical landscape data as input. The code is composed of a simulation engine that may be integrated into other scientific environments ranging from SciPy/Numpy to Fortran numerical weather models [Filippi *et al.(2013)*]. Validation cannot be strictly achieved for such phenomenon, but simulation has been tested in various scenarios on Mediterranean landscape in [Filippi *et al.(2014)*] and [Santoni *et al.(2011)*]. This paper briefly presents first the numerical method that is used in the solver and in the last section the different components and uses cases of the simulation code.

2 Wildfire propagation simulator method

ForeFire belongs to the family of fire area simulators, such as FarSite [Finney(2004)] for which the fire font is defined as the contour of the burning area. This contour expands in each of its points along its local normal with a prescribed velocity. ForeFire differs mainly in three points from available simulators :

In: G. Di Stefano, A. Navarra Editors: Proceedings of the RSFF'18 Workshop, L'Aquila, Italy, 19-20-July-2018, published at http://ceur-ws.org

1. the code can be run standalone or coupled with an atmospheric model, with similar and simple initialization procedures,

2. the numerical method used to track the fire front that relies on a constant CFL asynchronous advection of markers,

3. Velocity model(s) prescribing the propagation models may be added as well as flux(es) models for diagnostic.

These specificities have been chosen at the design stage of the code with the simulation of large wildfire in mind and maximum flexibility in terms of physical parametrization. It implies a particular processing of the fire front propagation and numerical physical diagnoses that can be extrapolated from the front dynamics.

2.1 Tracking the fire front propagation

Though usually resolved by Eulerian methods in the literature such as the level-set method, this family of method (called front-capturing as the front is captured according to the values taken by a 'marker' field) has proved to be computationally expansive as it simulates the spatial evolution of the state of the system in the whole domain [Maitre(2006)].

Front-tracking methods [Karimabadi *et al.(2005)*] on the contrary are designed to simulate only the spatial evolution of the interface and not a state evolution on the whole domain. The interface is described as a set of markers or vertices with coordinates that can evolve continuously in the domain (as opposed to fixed grids coordinates or particles in cell as [Coen and Schroeder(2013)]). By dealing only with the frontier, front-tracking methods need to discretize objects of smaller dimension than front-capturing methods. For example in wildland fire the discretization of the fire front is carried by approximating only the fire front line which is a 1D object, whereas front-capturing methods would have to approximate a 2D surface. Moreover the avection of each marker is relatively easy and computationally efficient way. The benefit in computational time is obvious but the counterpart is the need to handle topological changes with care. The objectives of the method used in ForeFire is to be as precise as possible to track the position of the fire front, deducing fuel loss, and requires to be less dependent on the input data resolution (wind, fuel), with less constraints on time-step.

2.1.1 Constant-CFL asynchronous advection of markers

Unlike conventional Lagrange methods for the advection of markers in a flow, Discrete Event Simulation (DES) advect its markers by solving the inverse problem, *i.e.* fixing a quantum distance rather than a time step. Instead of computing the spatial displacement while moving towards its new location in time $t + dt$ the proposed algorithm computes the duration taken by a marker to move towards its next location.

This paradigm for solving advection results in a continuous discretization of time, *i.e.* $t_i^{(n)}$ can take arbitrary values. This means that:

1. markers will not move at adjacent times, *i.e.* a record of which markers are to be advected according to their final time $t_i^{(n+1)}$ has to be handled,

2. advection for markers with a large velocity is processed much more often than markers with slow motion.

This feature is of great interest in strongly inhomogeneous flows such as wildland fires as it focuses the computation on the areas of high velocity, which usually corresponds to the areas of interest. Indeed the markers with low velocities will have large time-steps and thus will only be re-computed in a large amount of time, leaving space for the computation of high velocity markers.

3 ForeFire Software Architecture

ForeFire assumes that all the input data is contained in a single NetCDF file (fuel, wind, domain). Generating this landscape file is part of companion scripts and may be done using either atmospheric models or simple raw station data. One the data input is available, simulation in itself may be performed in various ways described in this section. Specifically, this is possible because the software compiles in a shared library, that has optional bindings to different programming languages and data formats to use as input/outputs (figure 1).

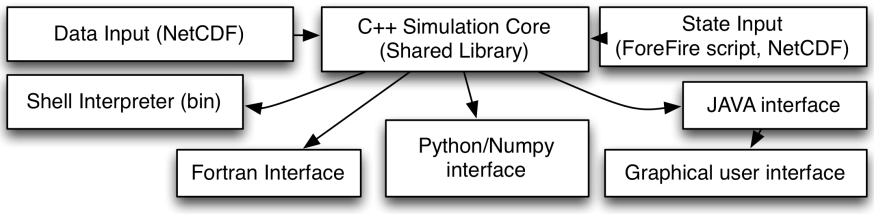

Figure 1: ForeFire modules

3.1 Command interaction

Interaction with the simulation is made with so-called commands, these can be either software calls (functions available in Fortran/Python/C++/Java interfaces) or command calls from the interpreter (a binary file launched from the command line) (figure 2).

```
setParameters[propagationModel=Iso;Iso.speed=1]
FireDomain[sw=(-10.,-10.,0.);ne=(10.,10.,0.);t=0.]
    FireFront[t=0.]
        FireNode[loc=(-3,-3,0.);vel=(-0.5,-0.3,0.);t=0.]
        FireNode[loc=(0.,3,0.);vel=(0.2,1.2,0.);t=0.]
        FireNode[loc=(3,0.,0.);vel=(0.7,0.1,0.);t=0.]
step[dt=10s]
print[sim1.ff]
clear[]
include[sim1.ff]
step[dt=5.2s]
quit[]
```

Figure 2: ForeFire scripting syntax

3.2 Adding new models

Adding a velocity model such as Balbi (Balbi et al, 2009) is simply done by adding a C++ file that defines a getSpeed function, a name, and a constructor to specify the required parameters (figure 3).

```
namespace libforefire {
/* model name */
const string ParametricROS::name = "WindSlope";
ParametricROS::ParametricROS(const int & mindex, DataBroker* db)
        : PropagationModel(mindex, db) {
    /* Locally interpolated values */
    effectiveSlope = registerProperty("effectiveSlope");
    normalWind = registerProperty("normalWind");
    /* Global parameters */
    windFactor = params->getDouble("WindAided.windFactor");
    slopeFactor = params->getDouble("WindAided.slopeFactor");
}

/*  Velocity function */
double ParametricROS::getSpeed(double* localValue){
    double speed = windFactor*localValue[normalWind]
            + slopeFactor*(1.+localValue[effectiveSlope]);
    if ( speed > 0. ) return speed;
    return 0;
}
}
```

Figure 3: C++ Propagation Model

3.3 Use Case: simple simulation in Python

The originality of the code is its ability to be used in a variety of contexts, so the same developments (on a flux or velocity model, dataset, algorithm...) may be used to be tested in a pure analytic research mode, make

its way to operations, and be compared easily with other approaches available from the same code. Python / Numpy bindings have been built to help C++ model development in a scientific environment (figure 4), with simulations that can benefit from all other Python bindings.

```
ff = forefire.PLibForeFire()
ff.setString("propagationModel
ff.setString("fuelsTableFile",
# Parameters for the global wi
ff.setDouble("velU",0.)
ff.setDouble("velV",12.)
# Domain Definition
ff.execute("FireDomain[sw=(0.,
# Adding layers
ff.addLayer("data","windU","ve
ff.addLayer("data","windV","ve
# Defining a specific fuel map
fuelmap = np.zeros((sizeX,size
fuelmap[:,90:110,:] = 1
fuelmap[150:,90:110,:] = 3
ff.addIndexLayer("table","fuel",0 , 0, 0, sizeX, sizeY, 0, fuelmap)
ff.execute("    FireFront[t=0.]")
ff.execute("        FireNode[loc=(40,60,0.);vel=(-1.,-1.,0.);t=0.]")
ff.execute("        FireNode[loc=(40,65,0.);vel=(-1.,1.,0.);t=0.]")
ff.execute("        FireNode[loc=(260,65,0.);vel=(1.,1.,0.);t=0.]")
ff.execute("        FireNode[loc=(260,60,0.);vel=(1.,-1.,0.);t=0.]")
# Storing 20 fronts
for i in range(1,20):
    ff.execute("goTo[t=%f]"%(i*20))
    pathes += printToPathe( ff.execute("print[]"))
# Plotting the fuel map
CS = ax.imshow(ff.getDoubleArray("fuel"))
# Plotting the firelines
for path in pathes:
    ax.add_patch(mpatches.PathPatch(path,edgecolor='red'))
plt.show()
```

Figure 4: Sample python ideal fire propagation and graphical output

3.4 Simulation in operational context

Here, user does not have to take time to prepare data, so everything is already available (a NetCDF file for each small region with all data layers). and the simulation is just run by using the loadData, then addFire, and print functions in a loop on the server and sent back to the web browser (figure 5).

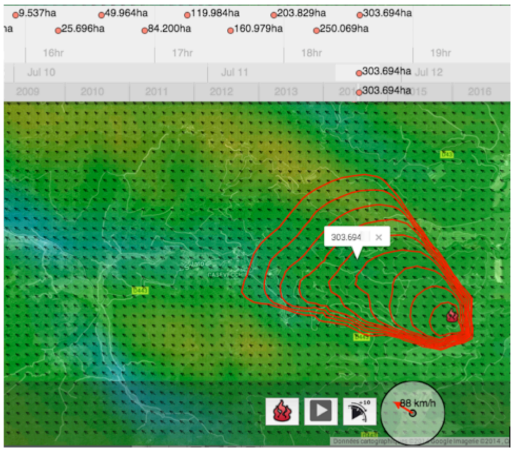

Figure 5: Simulation on real landscape with web Interface

4 Conclusion

The philosophy of the tool presented here is typical of both scientific and operational software, with sources available, self compiling and expandable code but also having connections with data formats that allows to simulate real fire on existing data in a very limited time. Further work is focused on coupling with real-time data feed, enhance the python interface and add more interaction methods.

Acknowledgements

This research is supported the Agence Nationale de la Recherche, grant ANR-16-CE04-0006 FireCaster.

References

[Finney(2004)] Finney, M. *FARSITE: Fire Area SimulatorModel Development and Evaluation—Research Paper RMRS-RP-4 Revised*; Technical Report; U.S. Department of Agriculture, Forest Service, Rocky Mountain Research Station: Ogden, UT, USA, 2004.

[Filippi *et al.(2011)]* Filippi, J.; Bosseur, F.; Pialat, X.; Santoni, P.; Strada, S.; Mari, C. Simulation of Coupled Fire/Atmosphere Interaction with the MesoNH-ForeFire Models. *J. Combust.* **2011**, *2011*, 1–13.

[Balbi *et al.(2009)]* Balbi, J.; Morandini, F.; Silvani, X.; Filippi, J.; Rinieri, F. A Physical Model for Wildland Fires. *Combust. Flame* **2009**, *156*, 2217–2230.

[Filippi *et al.(2009)]* Filippi, J.; Morandini, F.; Balbi, J.; Hill, D. Discrete Event Front-tracking Simulation of a Physical Fire-spread Model. *Simulation* **2009**, *86*, 629–646.

[Filippi *et al.(2009)]* Filippi, J.; Bosseur, F.; Mari, C.; Lac, C.; Moigne, P.L.; Cuenot, B.; Veynante, D.; Cariolle, D.; Balbi, J. Coupled atmosphere–wildland fire modelling. *J. Adv. Model. Earth Syst.* **2009**, *1*, doi:10.3894/JAMES.2009.1.11.

[Filippi *et al.(2013)]* Filippi, J.; Pialat, X.; Clements, C. Assessment of ForeFire/Meso-NH for wildland fire/atmosphere coupled simulation of the FireFlux experiment. *Proc. Combust. Inst.* **2013**, *34*, 2633–2640.

[Maitre(2006)] Maitre, E. Review of numerical methods for free interfaces. Ecole Thématique "Modèles de champ de phase pour l'évolution de structures complexes". Les Houches **2006**, *27*, 31.

[Filippi *et al.(2014)]* Filippi, J.; Mallet, V.; Nader, B. Evaluation of forest fire models on a large observation database. *Nat. Hazards Earth Syst. Sci.* **2014**, *14*, 3077–3091.

[Karimabadi *et al.(2005)]* Karimabadi, H.; Driscoll, J.; Omelchenko, Y.; Omidi, N. A new asynchronous methodology for modeling of physical systems: breaking the curse of courant condition. *J. Comput. Phys.* **2005**, *205*, 755–775.

[Santoni *et al.(2011)]* Santoni, P.; Filippi, J.; Balbi, J.; Bosseur, F. Wildland Fire Behaviour Case Studies and Fuel Models for Landscape-Scale Fire Modeling. *J. Combust.* **2011**, *2011*, 613424.

[Coen and Schroeder(2013)] Coen, J.; Schroeder, W. Use of spatially refined satellite remote sensing fire detection data to initialize and evaluate coupled weather-wildfire growth model simulations. *Geophys. Res. Lett.* **2013**, *40*, 5536–5541.