University of California Berkeley

Computer Science Department

CS 267

Project Report: IGNIS— A stochastic fire management simulator

Submitted by
Cristobal Pais
Alexander Wu
CS 267
May 8th
Spring 2018

Contents

Project Report: IGNIS— A stochastic fire management simulator

C.Pais, A.Wu

May 8th 2018

Abstract

IGNIS is a stochastic cell-based fire spread simulator. It includes a wide range of options that gives flexibility to the user when performing simulations such as statistical analysis, graphical outputs, and GIS interaction. Incorporating existing fire models from the Canadian Fire Behavior Prediction (FPB) System for its spread dynamics, IGNIS is designed for assessing the impact of harvesting designated forest stands on landscape flammability and expected losses in a heterogeneous landscape. Parallel implementations in C++ are developed and tested for real large-scale simulations using OpenMP, MPI, and CUDA GPU approaches. Best results are achieved using OpenMP API, obtaining strong-scaling and weak-scaling efficiency factors up to 80% and speed-ups up to 20x with 32 threads. Real fire instances are tested and compared to existing simulators outputs for performance comparison. Future extensions are discussed.

1 Problem description

The effects of global warming on temperature, precipitation levels, and soil moisture have increased the number of forest fires in different places of the world ([5], [8], [9]). Wildfires have consumed large areas in recent years, fire-fighting expenditures by federal land-management agencies have increased, hundreds of homes are burned annually, and damages to natural resources have been irreversible. Examples of this are the recent catastrophic incidents in Oklahoma, Chile, Portugal and southwestern Australia in the years 2016-2017. Considerable efforts have been made in this regard, however, a wildfire is a complex and difficult to model process.

The most important parameters for understanding a wildfire are the fire intensity and rate of spread (ROS). The fire intensity determines the scorch height and thereby the amount of the plant canopy consumed, killed or unburnt, and the ROS determines the time periods for which plants and animals will be subjected to lethally high temperatures [10]. In addition, variables such as fuel moisture, fuel loading, wind velocity, relative humidity, slope, and solar radiation are all recognized as producing important effects on fire. An important system that includes the aspects above is the Canadian Forest Fire Behavior Prediction (FBP) System. The FBP is a fire behavior model that predicts the rate of spread and the intensity of wildfire based on fuel, weather, and topographical inputs for each individual cell (forest units of approximately 100×100 sq. meters) inside a forest/grid [2]. However, it is not capable of predicting how a fire will progress through a heterogeneous landscape/grid over time. To overcome this, Prometheus, a deterministic fire growth simulator was released in May 2009 [7]. Using spatial fire behavior input data on topography (slope, aspect, and elevation) and FBP fuel types along with a weather stream, it simulates fire growth based on Huygens' principle of wave propagation, i.e., the fire front propagation is calculated in a fashion similar to a wave, shifting and moving continuously in time and space.

Another featured simulator is FARSITE [3] based on the American BEHAVE System. As indicated in [6], the two models found to best simulate the historical fires were vector-based implementations: FARSITE in the United States and Prometheus in Canada. A review of twenty-three simulators applicable to forecasting forest fire propagation can be found in [4]. Although the increasing interest in developing detailed cell-based deterministic/stochastic fire simulators for improving the accuracy of the simulations have been growing for the last years, a series of simplifications such as in [1] have been included due to the scale of the problem: memoryless distributions (Markovian processes) for modeling the fire spread dynamic, homogeneous forests (cells' characteristics are identical), reduced number of adjacent cells from 8 to 6 or 4, no spotting (fire

"jump" effect), no stochasticity is included, and wave/elliptic shape models are used as approximations for cell-based systems. We want to incorporate all complex details of a real system in IGNIS - a parallel cell-based fire simulator -, aiming to obtain high performance when simulating large-scale instances while providing relevant outputs for the forestry managers. Thus, parallel implementations following classic frameworks such as OpenMP, MPI, and CUDA GPU are tested, simulating real instances by processing all the cells' fire spread dynamic in parallel threads/processes, a problem that has not been successfully addressed to date.

Furthermore, in spite of the existence of several deterministic and stochastic fire simulators, smart-fire fuel management requires that both the harvest and simulation model have a well-structured interface for easily exchanging data at each iteration of the process. The above-mentioned simulators were not designed for purposes other than simulation, failing to be included into a fuel management framework. IGNIS is designed specifically for using it in a fuel-management procedure with the intent to mitigate the impact of large detrimental fires efficiently. Thus, IGNIS could work as a pure simulation tool, and/or as a tool that includes fuel-management decisions as well as a compatible tool with optimization software.

1.1 General Objective

The main objective of the project is to develop an efficient and realistic fire spread simulator that allows the researcher to simulate the fire dynamics inside a grid instance representing a real forest - or user-generated instance - based on variables such as weather, fuel type of each cell, elevation (terrain components), ignition points, and spotting probabilities, given an initial ignition area or lightning model. Future integration, evaluation, and design of harvesting management plans and fire spread control models would be implemented.

2 Methodology

The high complexity of the simulation scheme arises as a challenge for measuring the performance of our implementation: each fire has a large number of parameters and specific characteristics that lead to different outcomes and thus, the potential for parallelizing. Thus, different fuel types, weather streams, forest structures, and/or ignition points could lead to completely different fire dynamics (number of simultaneous burning cells, number of burnt hectares, etc.), and hence, to different serial/parallel performance. In order to account this situation, multiple instances/forests based on real data are generated, averaging their results, and comparing the performance of our code with respect to the serial implementation.

Performance is measured by calculating both the strong and weak scaling efficiencies - as well as speedup factors - obtained for different experimental instances ranging from sizes (number of cells inside the forest) $n \in [4 - 100M]$. A series of plots are generated for each experiment in order to visualize the performance of our parallel implementation across the generated and real instances. In addition, the accuracy of the simulations is contrasted with real fire scars - real instances, as well as state-of-the-art simulators (Prometheus).

2.1 Instances: Data

Two general weather files from weather stations located in Canada containing all relevant inputs for 7 and 36 hours are used in all instances. Each set of experiments for a size n uses the same ignition points for comparison purposes, starting the fire dynamic at the same hour for 1-minute precision clock step-size. Larger instances are generated by mixing locations from real data (see Appendix section 7.2) gathered from Canadian forests. In addition, a homogeneous instance (same fuel type for all cells) is included on each experimental set for comparison purposes.

2.2 Hardware & Software

The optimization and parallelization of IGNIS are developed for a specific hardware and runtime environment from The National Energy Research Scientific Computing Center (NERSC). In addition, tests have been

performed in a common daily-use laptop for comparison purposes. All experiments, benchmarks, and performance results are implemented using the following hardware and software:

1. **NERSC's Cori supercomputer: Phase I**
   - Intel® Xeon™ Processor E5-2698 v3 ("Haswell") at 2.3 GHz (32 cores per node)
   - 64 KB 8-way L1, 256 KB 8-way set L2, and 40 MB 20-way set L3 cache (shared per socket)
   - SUSE Linux version 4.4.74-92.38-default. Built with g++ version 4.8.5
2. **Daily-use laptop**
   - Intel® Core I7 4510U at 2.0 GHz (2 cores)
   - 64 KB 8-way set L1, 2 x 256 KB 8-way set L2, and 4 MB 16-way set L3 cache
   - Ubuntu 16.04.2 LTS / Windows 10

3 Solution Description

In this section, we describe the main strategies applied for modeling the problem in an efficient and effective way, as well as the modifications performed for parallelizing and optimizing our initial prototype.

3.1 Prototype: Python

Initially, a prototype version has been developed in Python exploiting its faster debugging/testing loop. Following an object-oriented programming, a series of classes for the most relevant components of the problem are developed: Cells, Weather, Forest, Ignitions, FBP methods, and I/O parsing. Then, the main program instantiates the different objects and apply the pertinent methods for performing the simulations.

The main simulation steps are the following:

i) Based on historical data, user provided, and/or a probability distribution, a cell will be ignited (or not). If the ignition is positive, the fire spread model will start to simulate its dynamics using the weather and forest information inside IGNIS.

ii) Relevant fire parameters are calculated by performing calls to the FBP module, obtaining the rate of spread (ROS) for each available axis of the burning cell: based on the weather, topography, and fuel characteristics, different ROS(θ,t) are obtained where θ is the angle w.r.t. the adjacent cells and t is the current period. Following a **discrete time** simulation approach, the internal simulator clock advances — a precision user-input parameter — and the fire progress is updated for each axis.

iii) The fire between cells is modeled as a **sending/receiving message** approach — ready for parallelization — based on the ROS values calculated for each axis. If the fire reaches the center of a cell, a message is sent. This step is reflected by a "*manageFire()*" method. Once all messages are collected from all burning cells, each receiving cell calls the "*getburned()*" method. Checking relevant environmental and its own conditions, a new ignition occurs (or not). This is the core of the simulator and thus, the critical performance bottleneck when simulating massive instances. However, its design has been structured for maximizing the parallel performance of the code, obtaining a large percentage of naturally parallelizable code, as we discuss in sections 3.2 and 4.

If provided by the user, a harvesting plan following an algorithm, heuristic or deterministic policy is implemented at certain periods of time (denoted by harvesting periods). Thus, forestry researchers can easily test their forest management plans by following a simple code template or via a .csv file.

iv) The previous step is repeated until a certain ending criterion is achieved: maximum number of weather periods, maximum simulation time, and/or burnt-out conditions (ROS minimum thresholds for fire spread, temperature, etc.). Relevant statistics regarding the status of the forest as well as (optional) plots and other outputs of the fire scar evolution are obtained.

Below, we can see an example of a forest with 9 cells where a fire ignites - potentially after a harvesting period - in cell 4 and then it spreads to different harvest units for two more periods. If no messages are sent to neighboring cells based on the current environmental conditions, end of the fire periods, go to next ignition or stop the simulation.

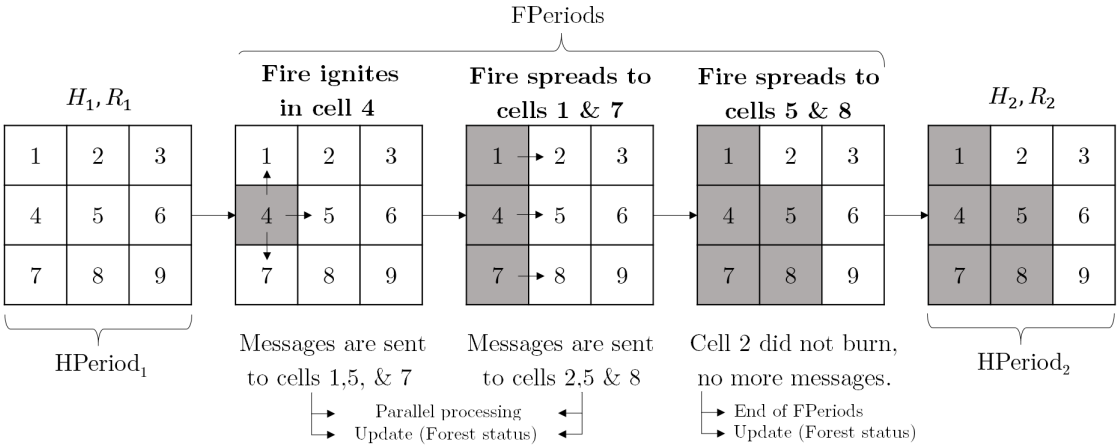


Figure 1: Simulation scheme: Send/Receive messages structure makes the problem highly parallelizable.

3.1.1 Main Inputs

The relevant inputs needed to perform a detailed simulation are the following:

- **Forest raster data:** ASCII grid forest files containing the number of cells, geographical coordinates, and information for each cell inside the forest such as fuel type, elevation, the percentage of slope, curving degree, slope azimuth, etc. Files in .csv or .asc (from GIS) format are valid.
- **Fuel types dictionary:** Fuel types codes and description, matching the nomenclature of the FBP module.
- **Ignition points:** Optional, a .csv file containing the cell(s) to ignite during the simulation.
- **Weather stream:** Hourly weather records from one or several stations located near the area of interest containing the date, average precipitation, temperature, wind speed/direction, humidity, as well as relevant environmental indexes associated with the fire dynamics, registered/approximated by the stations and equations inside the FBP module.

3.1.2 Main Outputs

- **Grids:** Simple .csv files with 1s indicating a burnt cell and 0s for available ones. Useful for statistical comparisons with other simulators as well as to generate fire probability maps.
- **Plots:** Fire scar evolution and sending/receiving messages dynamic can be visualized by a series of plots generated after the simulation.
- **Statistics (multiple replications):** Average number of burnt cells, the average percentage of the forest available, the benefit obtained - if harvesting plan is supplied.
- **Probability maps:** Empirical burn probability maps based on the output of a series of simulations.
- **3D terrain animation:** Grid files are translated into a 3D terrain mesh generated via satellite pictures and Google maps textures, obtaining a full animation of the fire evolution (work in progress).

3.1.3 Main Parameters

- **Head, Flank, and Back ROS factors:** Rate of spreads to all 4 main directions. A simple linear distribution procedure is performed to obtain the other 4 directions.
- **Weather period length:** Duration of each weather period (1 hour by default).
- **Fire period length:** Step size of the internal clock of the simulator. Larger values give an approximation of the real fire while sacrificing accuracy (same as weather period by default).
- **Options:** A series of options such as generate plots, grids, fire scenarios for optimization models, and/or statistics sheets are passed as arguments to IGNIS via command line.

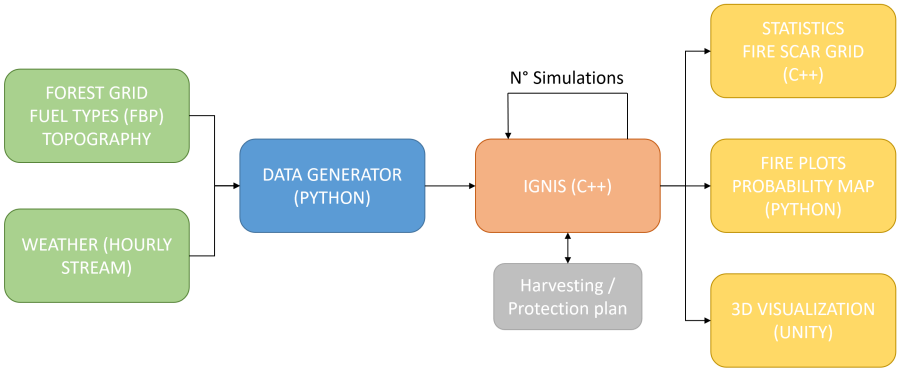


Figure 2: Simulation framework scheme.

3.2 C++ Serial version & Optimizations

Our next step after writing the Python prototype was to translate it into a lower-level language for performance. We chose C++ because the objects we created in Python for the most part naturally translate to C++ objects. In addition, after analyzing the Python code, we found that the sparse set of "unburned" neighbors of each Cell object was being naively updated. That is, at every time step, we check every cell and its corresponding neighbors for updates to the sparse set. Rather than do this, we decide to only update neighbors when a cell is newly burned.

With these two optimizations, as well as testing a series of compiler's flags, we achieved 15-20x speedups in performance, depending on the simulation that was run.

3.2.1 Parallel Structure

Our algorithm contained 3 sections at each time step: (1) checking for new lightning ignitions (igniting), (2) updating the intensity of already-burnt cells and analyzing newly burnt ones (sending messages), and (3) marking newly burnt cells as burning (receiving messages).

The ignition stage is very quick (less than 1% of total execution time), with most simulations only igniting a single time at the first time step of the simulation. The sending messages stage updates the intensity of the fire in every burning cell. Because we can have a large number of cells burning at once, and there are no direct dependencies on neighboring cells, this part is easily parallelizable. Each cell, in addition to updating itself, can also "send a burning message" to an adjacent cell. In the receiving messages stage, we analyze the "burn messages" sent to non-burning cells and mark them as burnt if the conditions are met. This part is also potentially parallelizable, but because the number of newly burnt cells at a single timestep is dwarfed by the number of currently burning cells, we found that a speedup here is of lower priority ($\approx 10\%$ of total execution time).

3.3 Parallelization: OpenMP work-sharing

Due to the well-designed and easily parallelizable structure of our code, the most suitable approach for parallelizing its execution consists of a shared-memory approach using the well-known OpenMP API. This is an advantage for our project since the code will be also optimized for its execution in normal desktop/laptop computers, without needing a multi-node architecture for exploiting its parallelism. In our parallel loop, we initially just added a work-sharing construct to split the iterations of the loops between our threads. We found that there were exactly two critical sections, in adding to two local data structures that accumulate 1) burn messages sent to neighboring cells, and 2) cells that were previously burning, but now burnt-out. We wrapped writes to those data structures in an OpenMP critical section.

3.3.1 Optimizations: Distributed Data structures

We found that we could easily make the loop embarrassingly parallel if instead of adding to a single data structure, we add to a data structure local to the worker thread. Since we would iterate over the initial data structure to reduce it into statistics anyways, we found the additional complexity of "distributing" the data structure does not scale with grid size. In addition, different loop scheduling options were tested: dynamic, guided, auto, runtime, and static, as well as the chunk-sized block process. Following a brute force optimization approach, we were able to obtain an average of 15%-20% extra performance for the parallel region.

One final improvement we made to our parallelism was analyzing the false-sharing effect. Because we had a $vector < DS >$ to store our "distributed" data structures, where DS is the data structure of choice, we found there to be a bottleneck on the parallelism exploited in the problem. Upon further analysis, we found this to be false-sharing in the array of DS backing the vector. After adding padding between elements of the array — where optimal values were obtained following a binary search optimization approach — we achieved a significant additional speedup from our previous attempt, as we discuss in section 4.

3.4 Parallelization: Other attempts

Besides the natural OpenMP implementation, we tested two parallel versions based on MPI and CUDA, for completeness, using the experience gathered during the semester.

3.4.1 MPI

We considered using MPI in a means similar to the grid division used in our particle simulation homework. However, after further thought and initial tests (both in Python/mpi4py and C++), we realized that the parallelism would be exceptionally poor because the fires would start out localized and spread from a single ignition point. This lack of uniformity throughout the grid would mean that our load balancing would be exceptionally poor. As a second approach, we considered the fact that having multiple ignition points could be easily parallelizable (up to the point of overlap between ignitions), but based on conversations with forestry researchers, the current project should be focused on single ignition points.

3.4.2 CUDA

We also tested CUDA to parallelize our inner loop, but we found that although our code does not have too many branches, one of the ROS-specific functions we call from the Canadian FBP library had many branches. This led our CUDA prototype to achieve poor speedup. Thus, no more than 5x speedup factors were obtained, achieving very limited performance. Hence, a complete re-design of the simulator methods and main objects would be needed to take full advantage of this parallel computing platform. Furthermore, the requirement of specific hardware components (Nvidia GPU units) limits the widespread parallel execution of IGNIS, generating a barrier for forestry researcher without access to this architecture, and thus, non-practical for the project.

4 Results & Discussion

In this section, we present the main results obtained from a series of experiments including the optimization techniques implemented. We focus our parallel analysis on the successful OpenMP implementation.

4.1 Serial version

4.1.1 Parallel section: % of total code

In order identify the potential benefits of a parallelization, we perform a detailed analysis of the execution times, breaking it into: (1) sending time ("parallel" region), (2) receiving time, (3) ignition time, and (4) copying time. In Tables 1 (a), (b) we can see that the average — across all instances — time spent in the parallelizable region represents a 79% of the total execution time. The other $\approx 20\%$ is divided evenly between (2) and (4), while (3) is almost negligible. These results give us a sense of the potential impact of an efficient parallel implementation in our code.

Instance (n)	AVG % Time in Parallel Zone: Sending		
	Bottom 10%	Middle 80%	Top 10%
4	80.44%	86.00%	94.10%
9	74.15%	82.63%	93.11%
400	56.75%	72.09%	95.84%
1600	64.32%	71.57%	90.45%
10K	54.00%	77.46%	88.34%
50K	62.00%	71.73%	87.43%
100K	64.50%	79.13%	91.23%
AVG	65.17%	77.23%	91.50%

Instance (n)	AVG % Time in Parallel Zone: Sending		
	Bottom 10%	Middle 80%	Top 10%
160K	72.43%	87.60%	88.32%
250K	70.82%	78.28%	90.05%
500K	69.94%	73.70%	80.19%
1M	68.64%	78.01%	89.00%
10M	60.25%	80.00%	92.43%
50M	58.30%	88.77%	93.11%
100M	52.00%	90.03%	95.24%
AVG	64.63%	82.34%	89.76%

Table 1: Average percentage of the running time parallelizable for different instances. Upper and lower tails are included for completeness. Results obtained by simulating 20 different forests (fuel types, spatial distribution, etc.) for each size n with the same weather conditions.

Important is to note the fact that some instances (lower tail) will experience a poor parallel performance. The reason behind this behavior is clear: certain combinations of fuel types and forests distributions lead to a significantly smaller set of simultaneous burning cells per simulation, and thus, the parallelization of (1) will not impact the overall execution times as much as we desire. Knowing this limitation, we proceed to analyze the solving times and accuracy of the simulations.

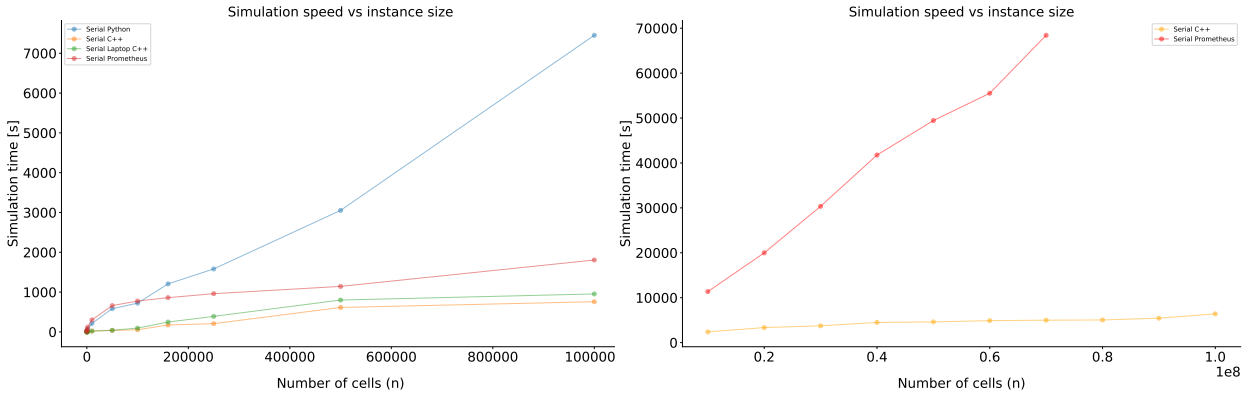
4.1.2 Running times

Comparing the running times of our pure Python prototype and C++ implementations with the results obtained using Prometheus, we can see in Figure 3 how the optimized version clearly outperforms the Python prototype, reaching up to 15-20x speedups when dealing with large instances. Furthermore, IGNIS (C++) obtains significant shorter times than Prometheus (up to 30x speedups). This is very important since wave-based simulators are performing a series of approximations when generating the final fire scar that simplifies the calculations performed under our cell-based approach, indicating that our implementation is efficient. Thus, we proceed to test the accuracy of IGNIS in the context of real fire instances.

Interesting is to notice the fact that Prometheus is not able to solve the three largest instances (80M, 90M, and 100M, out of memory error) mainly due to the way it is dealing with the cells: instead of only instantiate active cells (as in IGNIS), it is loading all the information in memory while updating and generating new fields as the simulation progress. Therefore, it is not suitable for massive instances.

4.1.3 Accuracy

We tested IGNIS with **7 real instances** from fires in Canada (see Appendix section 7.2) as well as a series of experimental instances generated from field data containing all the relevant inputs for the FBP



(a) Running times small instances. (b) Running times large instances

Figure 3: Running times for serial versions. Python's large instances results are omitted for visualization purposes. Large instances solved by parts (hours) to avoid user's max time limitations in Cori.

system. We compared IGNIS with state-of-the-art simulators such as Prometheus and the real fire scars based on real data. As an example, simulation results obtained by IGNIS and the real fire scar in the Dogrib region, Canada, are shown on the left and center of the picture below, obtaining **less than 10% of error** without tuning its default parameters in a fraction (1/20) of the time needed by Prometheus. An average error of 5.24% (correlation and image processing distance metrics) is obtained among all tested instances, supporting the design of IGNIS. As an extra, a 3D terrain (see Appendix 7.3) is generated based on satellite and topographical information, obtaining a 3D animation for visualization purposes as well as a burnt probability analysis.

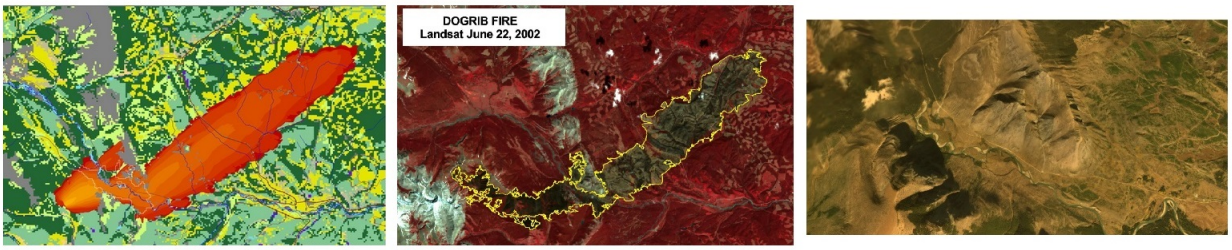


Figure 4: IGNIS' fire scar obtained for the region of Dogrib, Canada comparison with the real fire in 2002 and its 3D terrain (top view).

Future stages of the project will be focused on tuning specific parameters (e.g. burning-out conditions) in order to obtain the best accuracy possible, leading to better results and real-life fire management strategies evaluations depending on the characteristics of the forest under study.

4.2 Parallel implementation

In this section, we compare the performance of our best parallel implementation in terms of execution times for a different number of threads running inside the same node (single-node analysis). Important is to remember that we present the average results for different sets of instances, varying the size n of the forest in order to address the inherent variance and complexity of the simulations for different forests' structures. Therefore, we discuss the validity of the obtained performance results, taking into account the inherent conditions of the problem. Both the naive and the optimized C++ OpenMP versions are analyzed and compared.

4.2.1 Running times & speedup analysis

Looking at Table 2 we can see that the detailed and average speedups obtained for the small instances with the optimized OpenMP version are very good, obtaining high-performance with a certain number of threads. Performance is even better when dealing with the large instances set, improving each average speedup up to an average of 16.48x when running 32 parallel threads.

Table 2: Speedup factors for small instances for different number of threads.

Instance (n)	2 threads	4 threads	8 threads	16 threads	24 threads	32 threads
4	0.50	0.40	1.00	1.00	1.50	6.00
9	0.71	2.50	5.00	3.33	5.00	5.00
400	3.01	3.26	3.69	3.66	3.66	3.69
1600	3.07	3.50	4.18	4.34	4.34	4.34
10000	2.24	2.81	3.64	4.84	6.12	8.70
50000	1.93	3.21	3.51	3.86	4.23	8.66
100000	1.83	3.72	4.26	4.35	5.71	8.38
160000	1.86	3.05	3.29	3.63	3.58	10.50
250000	1.85	3.21	4.76	5.52	9.12	11.98
500000	1.98	3.71	6.44	8.88	13.06	17.45
1000000	2.10	3.67	6.09	8.99	12.16	17.13
AVG Speedup (OPT)	1.92	3.00	4.17	4.76	6.23	9.26

As expected, even better speedups are obtained when the dealing with homogeneous forests as can be seen in the Table 3 where a summary for the large instances speedup averages is shown. From this, we can see a near optimal average speedup up to 16 threads, while reaching a great $\approx 20x$ with 32 threads.

Table 3: Average speedups for large instances: heterogeneous and homogeneous forests

	2 threads	4 threads	8 threads	16 threads	24 threads	32 threads
AVG Speedup Large Homogeneous (OPT)	1.99	3.76	7.01	12.33	15.34	19.78
AVG Speedup Large Heterogeneous (OPT)	1.84	2.44	4.22	6.89	11.62	16.48

4.2.2 Strong Scaling

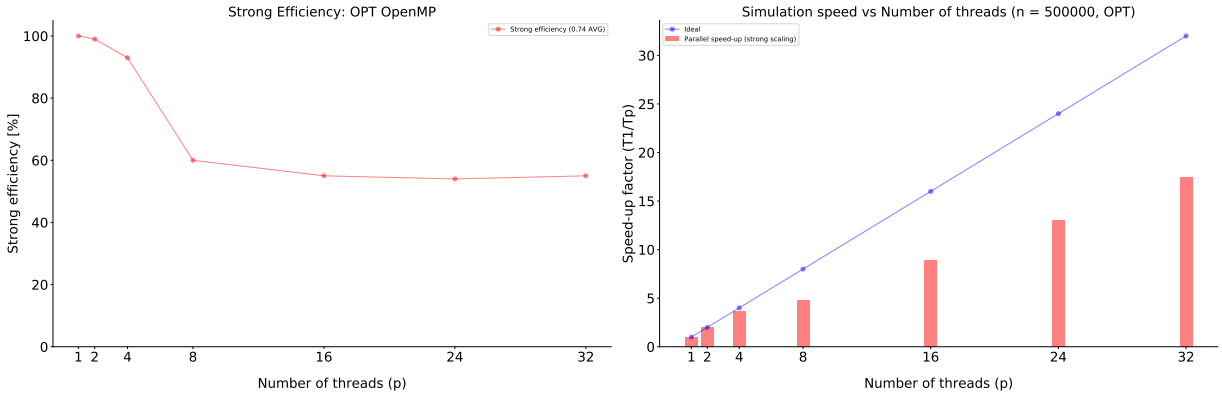
After generating the speedup and strong-scaling efficiency plots for the experimental instances, we can easily check our initial expectations: speedup factors, as well as strong-scaling efficiencies, are low with respect to the ideal benchmarks when using the naive OpenMP implementation, obtaining speedup factors below $10x$ and an average strong efficiency around 60%, while the optimized implementation is able to improve these numbers obtaining up to 15x and 20x speedups for the small and large instances, respectively, as well as averages strong efficiency factors between 75% and 82%, depending on the size and structure of the forest.

In Figures 5 (a) and (b) we present the results obtained for the average values obtained among 20 instances with 500,000 cells (representative research size case) using the optimized OpenMP implementation. Similar — slightly better — results are obtained for larger instances.

Based on all our experiments, adding more threads leads to better execution times following a flat pattern w.r.t. the strong-scaling efficiency. Thus, our optimized implementation is able to obtain a great strong scaling performance, taking into account the high complexity of the instances and variability of the results depending on the forest's structure.

4.2.3 Weak Scaling

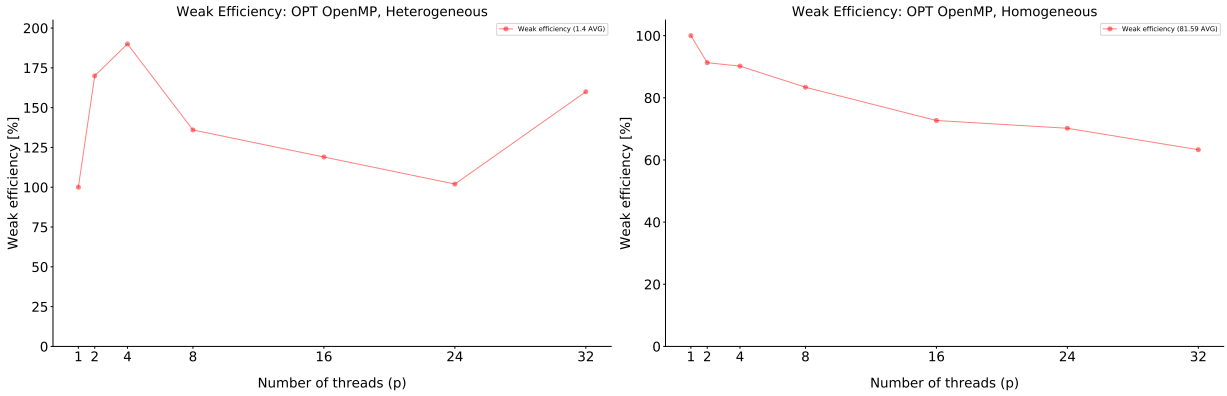
Due to the high dependency of the results w.r.t. the forest structure, we divide our analysis for heterogeneous and homogeneous forests. When the forests are heterogeneous, the comparison between instances of different sizes and number of threads looses its meaning since there is no guarantee that the problem will scale in



(a) Strong-scaling efficiency). (b) Speedup Factors

Figure 5: Strong-scaling and speedup factors for OPT version $n = 500,000$.

complexity: increasing the problem size doesn't directly affect the computations time, it significantly depends on the composition of the forest, leading to different fire dynamics. Therefore, we expect an erratic pattern when dealing with heterogeneous forests in terms of weak-scaling efficiency.



(a) Weak-scaling efficiency (Heterogeneous). (b) Weak-scaling efficiency (Homogeneous)

Figure 6: Weak-scaling and speedup factors for heterogeneous and homogeneous instances, starting with $n = 500,000$.

In Figures 6 (a) and (b) we can see the weak efficiency obtained for heterogeneous and homogeneous instances. As expected, results for heterogeneous instances lack of meaning since we are not comparing the same fire dynamics (and thus, the number of simultaneous burning cells, critical for the parallel performance). On the other hand, comparing homogeneous instances gives us correct results, since we compare the same fire dynamics reaching an average weak-scaling factor value equal to 81.6%. Again, similar and even better weak factors are obtained with larger (and homogeneous) instances, following the discussion above.

Therefore, our optimized parallel implementation is able to obtain high-performance values in both strong and weak scaling factors thanks to its natural and embarrassingly parallel design.

4.2.4 Summary

For our results, our "parallelism" is generally underreported: as we reported earlier, the parallelizable region lies between 60% and 90% of the overall runtime of our serial C++ implementation, therefore for many of our simulations, we can only achieve 3-10x parallelism in the ideal case.

Parallelism is easily exploitable when the burning areas are large, but there exists a significant instance-dependence performance: for forests with a generally nonburnable terrain, fire spread can be slow, at times causing neighboring cells to be nearly burnt out before it spreads to neighboring cells. This greatly mitigates

the amount of parallelism we can achieve, because the set of burning cells will not dramatically increase in size. Additional factors such as rivers, wind, and ignition location greatly affect our outputs.

To test the "maximum" parallelism of our simulation, we use a homogeneous forest of similar, burnable terrain, and find that the parallelism achieved is nearly ideal (for the "send messages" section we parallelized). However, in the above cases, where there inherently exists less parallelism, we are shown lower scaling numbers, when in fact we have exploited all of the (sparse) parallelism presented to us in the problem.

However, results are satisfactory, achieving over 100x performance relative to our initial Python prototype, and at least 20x performance over Prometheus and other simulators. Following extensions/optimizations of IGNIS could look towards parallelizing the other 10-20% of the code that we did not attempt. This will be particularly useful when adding aerial fire dynamics (complementary to spotting) such that a cell could spread its fire not only to its (at most) 8 adjacent cells but to a certain area determined by the environmental conditions and fire intensity.

Finally, important is to highlight the accuracy of IGNIS when simulating real instances such as Dogrib. As part of an ongoing research project, IGNIS will be implemented, tested, and used in practice in Chile, Spain, Canada, and the U.S. (California), and thus, its simulation performance is one of the most critical aspects of its development.

5 Conclusions & Future Work

- A successful serial and parallel implementations for a stochastic cells-based simulator have been developed. Obtained fire scars match state-of-the-art simulators' outputs (less than %10 of error in average) using default configurations.
- Running times obtained for the serial and parallel versions outperform state-of-the-art simulators by speedup factors up to 30x (serial) and over 100x (parallel).
- In order to obtain the best performance of parallel OpenMP approach, we needed to apply code optimization techniques as well as tuning relevant parameter involved in the parallelization of the code. Significant improvements in all performance metrics were obtained thanks to this step.
- Parallel implementations are able to reach average strong and weak scaling efficiencies — among different instance sizes and types — around 80% for single-node tests with $p \in \{1, 2, 4, 8, 16, 24, 32\}$ (p = number of threads) showing the great performance reached thanks to the embarrassingly parallel design of IGNIS.
- As a future step, the project will address the development of Fuel Prediction Behavior systems for the U.S., Chile, and European Forests based on national field data (work in progress).
- Full integration with GIS and 3D visualization module, as well as full integration with fuel-management heuristics component.
- A careful MPI implementation taking into account the difficult imbalance challenge would be useful for gaining strong-scaling performance, in particular when dealing with multiple ignition points and the potential overlap of multiple fire scars inside the same forest.
- The presence of several branches inside the FBP library prevented us to obtain an efficient implementation of a CUDA GPU parallel algorithm without re-designing the structure of the simulator and FBP functions. Therefore, a specific IGNIS version for this framework could be developed in the future in order to take advantage of its massive parallelism, where the most natural approach consists of associating cells to individual threads.
- A graphical user interface (GUI) should be developed in the future in order to reach non-technical users with the aim to obtain a friendly-user software package that can be effectively used by researchers of different areas.

6 References

1. Boychuk, D., Braun, W. J., Kulperger, R. J., Kroughly, Z. L., & Stanford, D. A. (2009). A stochastic fire growth model. Environmental and Ecological Statistics, 16, 133–151
2. Canadian Forest Fire Behavior Prediction (FBP) System: user's guide. 1996. Hirsch, K.G. Natural Resources Canada, Canadian Forest Service, Northern Forestry Centre, Edmonton, Alberta. Special Report 7. 122 p.
3. Finney, M. A. (2005). The challenge of quantitative risk analysis for wildland fire. Forest Ecology and Management, 211, 97–108.
4. Papadopoulos, G.D. and F.-N. Pavlidou. 2011. A comparative review on wild re simulators. IEEE Syst. J. 5: 233–243.
5. Running, S. M. (2006). Is global warming causing more large wildfires? Science 313, 927–928. (doi:10.1126/science. 1130370)
6. Sullivan, A. L. (2009). Wildland surface fire spread modelling, 1990–2007. 3: Simulation and mathematical analogue models. International Journal of Wildland Fire, 18(4), 387-403.
7. Tymstra C, Bryce RW, Wotton BM, Taylor SW, Armitage OB. 2010. Development and structure of Prometheus: the Canadian Wildland Fire Growth Simulation Model. Information Report NOR-X417.Edmonton(AB):Natural Resources Canada, Canadian Forest Service, Northern Forestry Centre. 102 p.
8. Westerling AL, Hidalgo HG, Cayan DR, Swetnam TW. (2006) Warming and earlier spring increase western US forest wildfire activity. Science 313, 940 – 943. (doi:10.1126/science.1128834)
9. Westerling ALR. 2016 Increasing western US forest wildfire activity: sensitivity to changes in the timing of spring. Phil. Trans. R. Soc. B 371: 20150178.
10. Whelan, R. J. (1995). The ecology of fire. Cambridge university press.

7 Appendix

7.1 Fire simulator dynamic

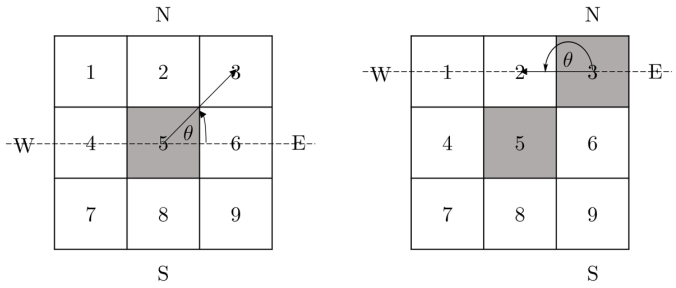


Figure 7: Adjacent cells simulator scheme and wind direction angle θ.

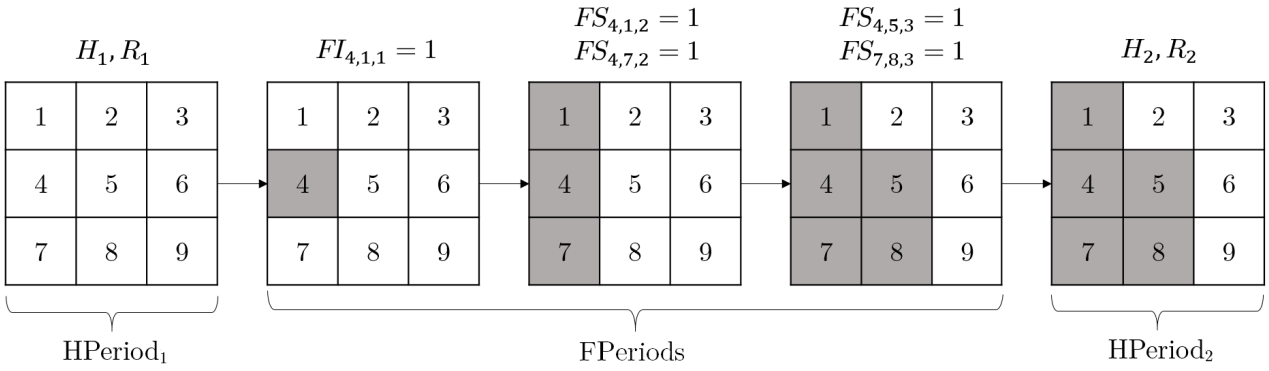


Figure 8: Simulation example: harvesting and fire periods are intercalated during a simulation period.

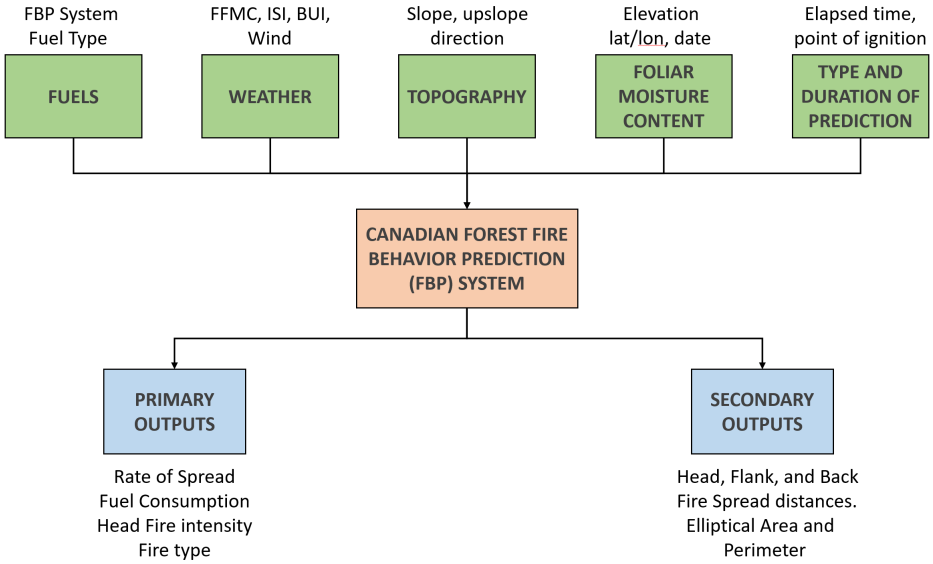


Figure 9: FBP system inputs/outputs scheme.

7.2 Real Instances

i) **Mica Creek**

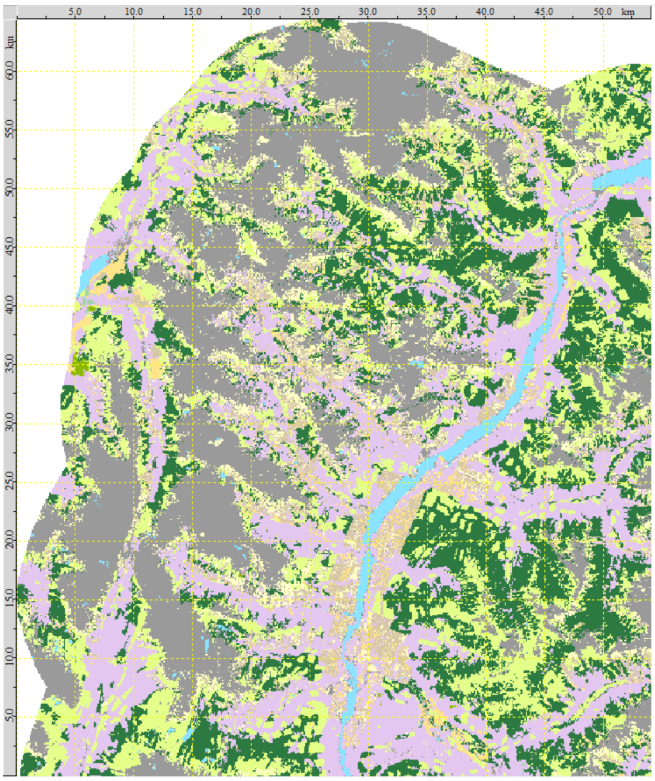


Figure 10: MicaCreek instance, Canada

ii) **Revelstoke**

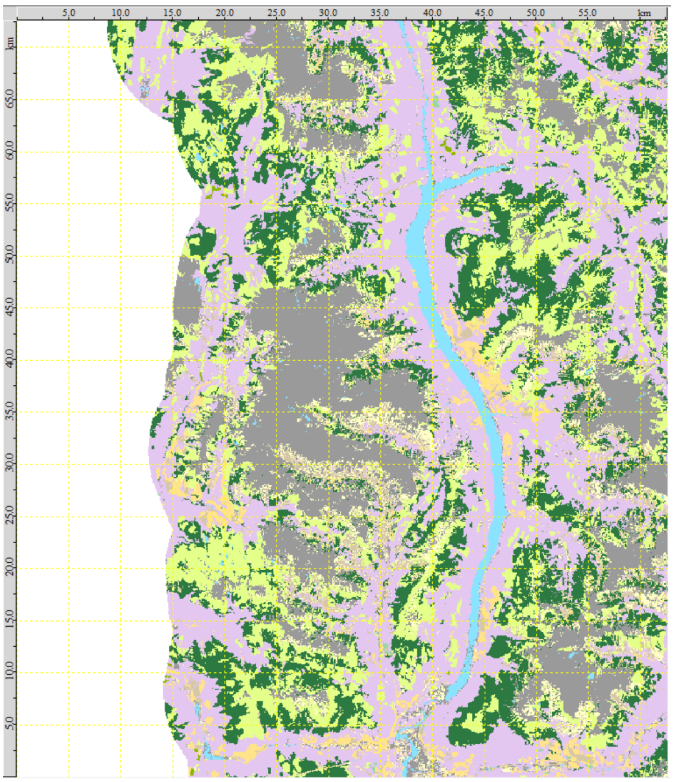


Figure 11: Revelstoke instance, Canada

iii) **Arrow**

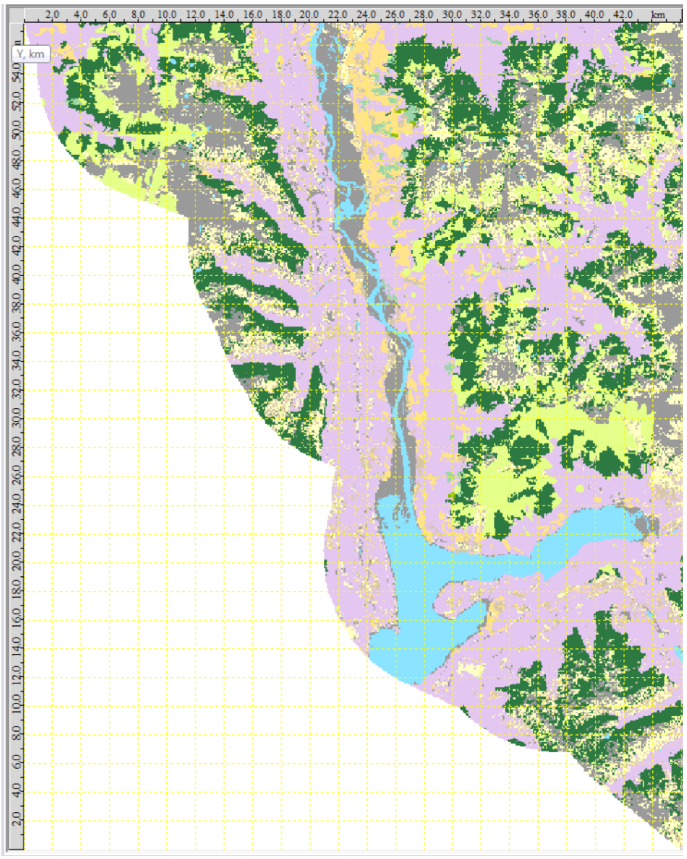


Figure 12: Arrow instance, Canada

iii) **Dogrib2D**

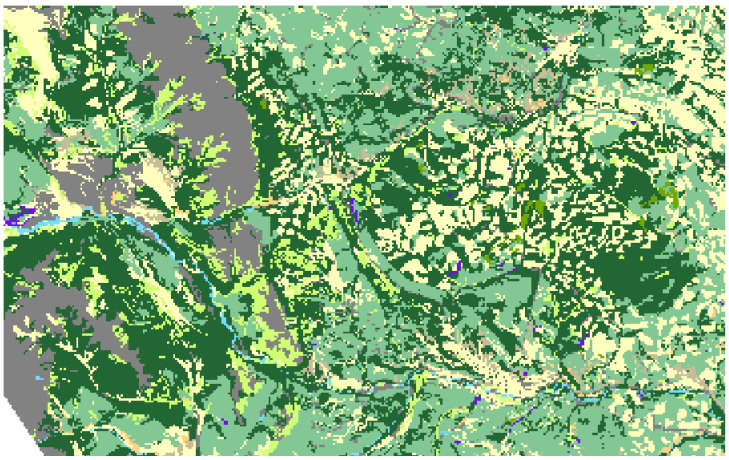


Figure 13: Dogrib instance, Canada

iv) **Central Kootenay**

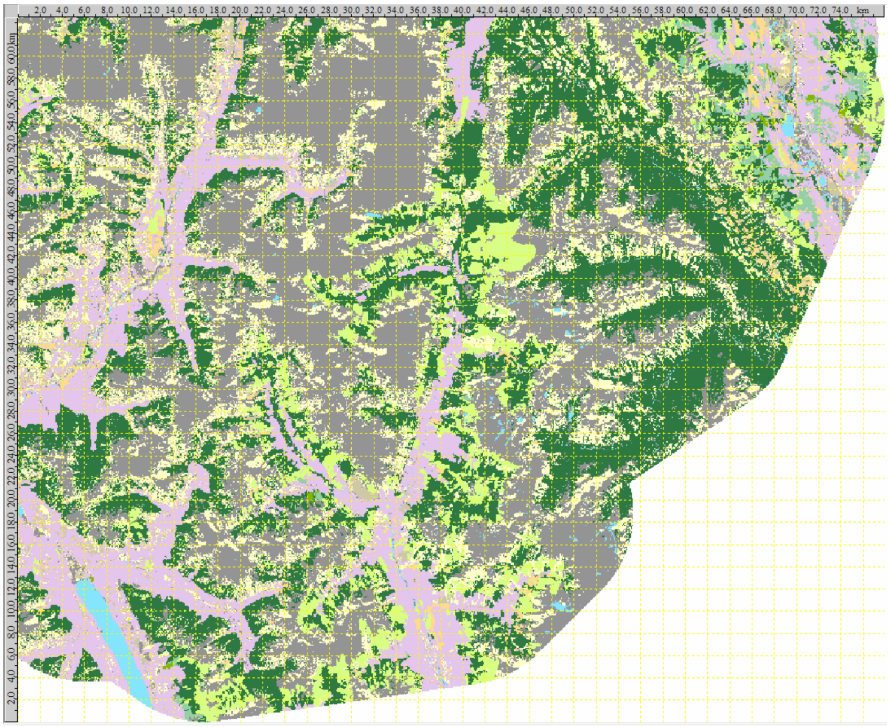


Figure 14: Central Kootenay instance, Canada

v) **Glacier National Park**

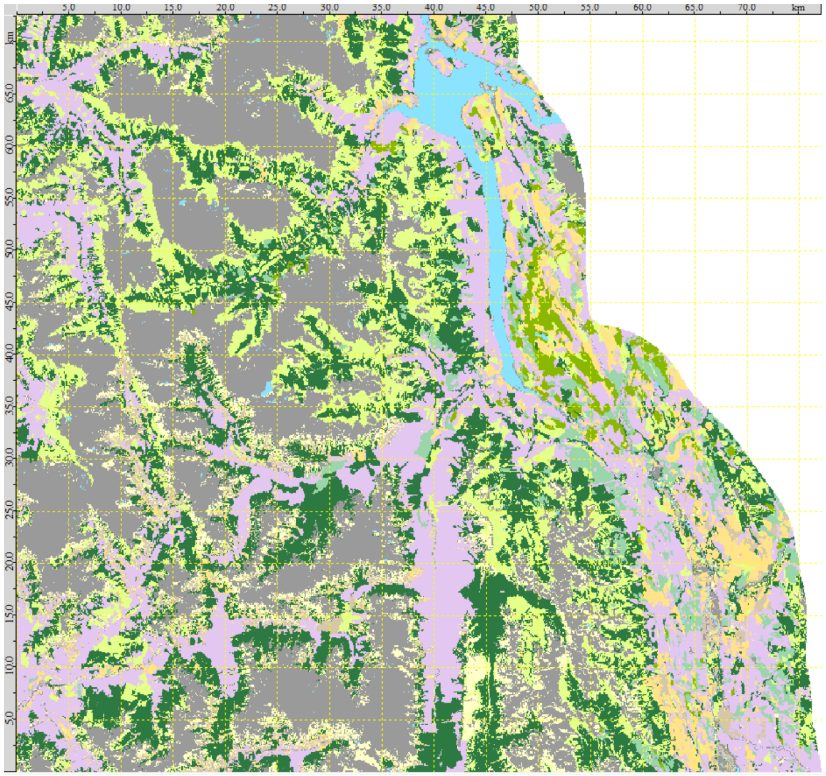


Figure 15: Central Glacier National Park instance, Canada

vi) **Neptune Peak**

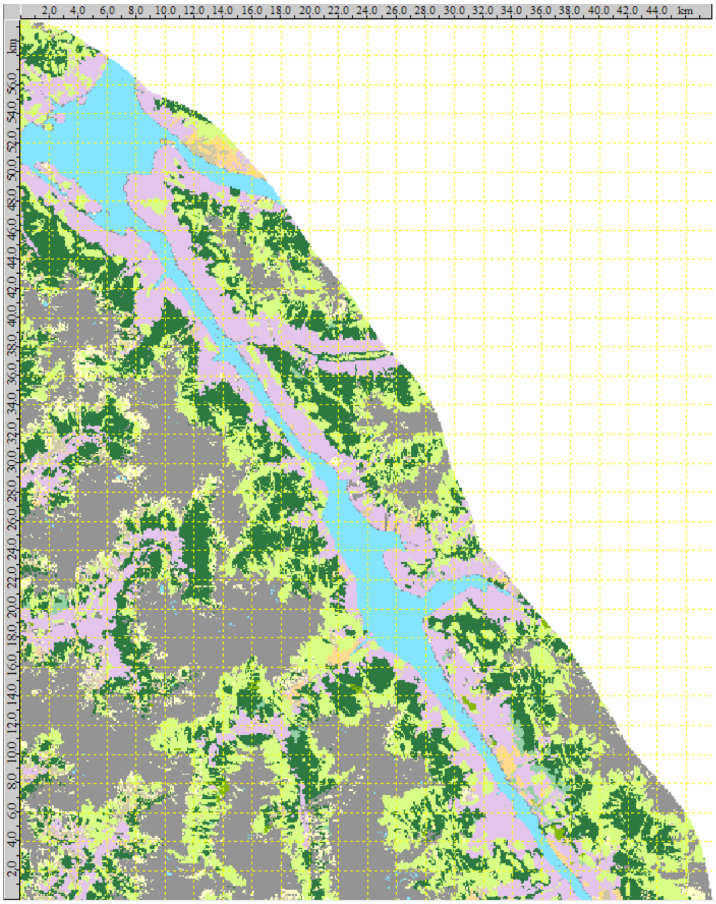


Figure 16: Neptune instance, Canada

7.3 3D Terrain generator

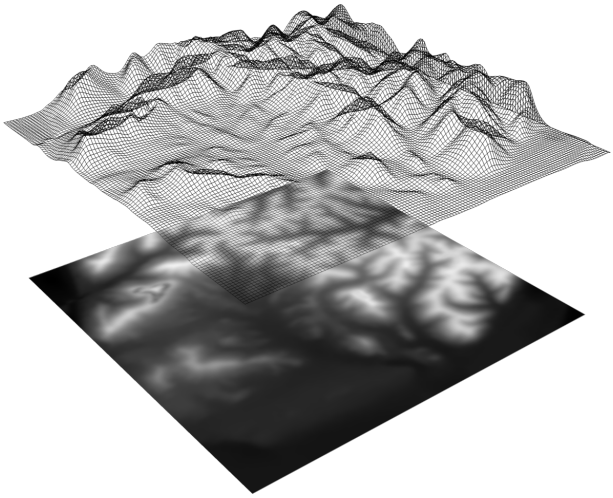


Figure 17: Heightmap to mesh transformation based on satellite information

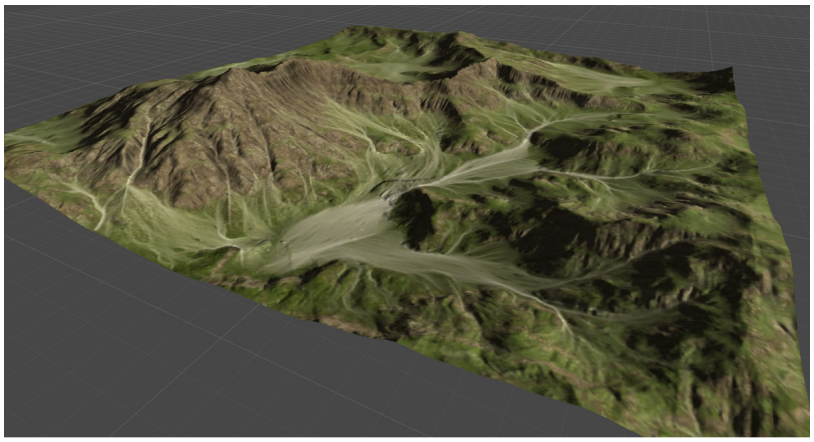


Figure 18: Textured map using Google maps database

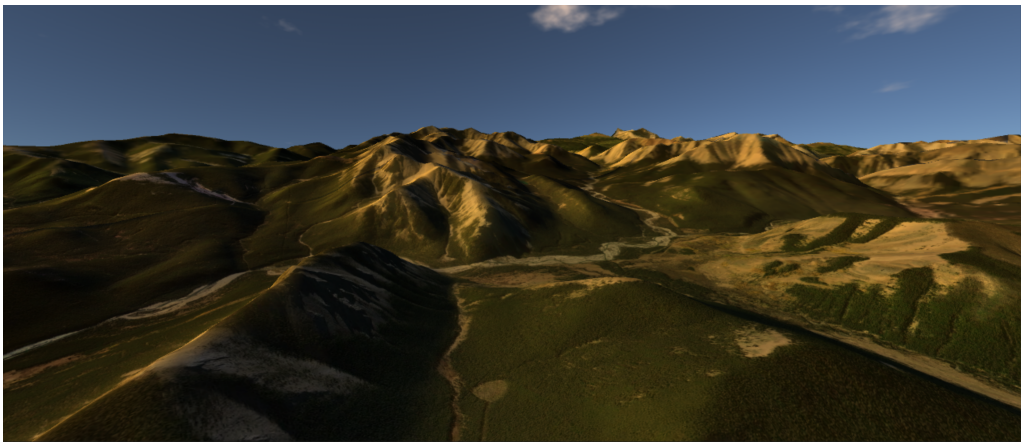


Figure 19: Dogrib 3D terrain generated from satellite information