University of Nevada, Reno

vFireLib: A Forest Fire Simulation Library Implemented on the GPU

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science
in Computer Science and Engineering

by

Jessica Elizabeth Smith

Dr. Frederick C. Harris, Jr., Thesis Advisor

May, 2016

THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

JESSICA SMITH

Entitled

vFireLib: A Forest Fire Simulation Library Implemented On The Gpu

be accepted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

Dr. Frederick C. Harris, Jr., Advisor

Dr. Sergiu M. Dascalu, Committee Member

Dr. Eric Wang, Graduate School Representative

David W. Zeh, Ph.D., Dean, Graduate School

May, 2016

Abstract

Forest fire simulation is a complex problem that requires an enormous amount of data processing. In order to operate a spread calculation in real-time, it becomes necessary to use parallel processing. Processing the spread calculations on the Graphics Processing Unit (GPU) allows hundreds of calculations to be performed at the same time, which allows the simulator to run up to 20X faster than its sequential counterpart. When the timings take up to half an hour, 20X faster makes the runtime feasable for a real-time application. This forest fire simulation library has the ability to incorporate base fire spread, fire acceleration, crowning, and a spotting prototype into the spread simulation. Three different methods for fire spread have been implemented to provide different perspectives to fire researchers.

Dedication

To Madeline, Melody and Lily, my mini future coding army.

Acknowledgments

I would like to thank Dr. Fred Harris, Dr. Sergiu Dascalu, and Dr. Eric Wang for being on my committee, with special thanks to Dr. Fred Harris for giving me the opportunity to do research in the HPCVIS lab. I would like to thank Dr. Roger Hoang for providing the original groundwork for this research. I would also like to thank Chase Carthen for his assistance in debugging my project even though he was not directly assigned to it, and his assistance with GDAL. I would also like to thank Thomas Rushton for helping me brainstorm and keeping me sane. I would like to thank all my other labmates for listening patiently as I bounced ideas off them.

This material is based in part upon work supported by: The National Science Foundation under grant number(s) IIA-1329469, and by Cubix Corporation through use of their PCIe slot expansion hardware solutions and HostEngine. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or Cubix Corporation.

Contents

Abstract **i**

Dedication **ii**

Acknowledgments **iii**

List of Tables **vi**

List of Figures **vii**

1 Introduction **1**

2 Background and Related Work **3**

2.1 Fire Models . . . 3
2.1.1 Overview . . . 3
2.1.2 Taxonomy of Fire . . . 4
2.1.3 Fire Shape . . . 4
2.1.4 Surface Fire . . . 5
2.1.5 Crowning . . . 7
2.1.6 Spotting . . . 8
2.2 Fire Simulators . . . 9
2.2.1 BEHAVE . . . 9
2.2.2 fireLib . . . 10
2.2.3 FARSITE . . . 10
2.2.4 vFire . . . 11
2.2.5 FIRETEC . . . 11
2.3 GPU Computation . . . 12
2.3.1 Overview . . . 12
2.3.2 CUDA . . . 12
2.4 Fuel Models . . . 14

3 Real-time GPU-based Wildfire Simulation **15**

4 Specification and Design **18**

4.1 Overview . . . 18
4.2 Service Requirements . . . 18

4.2.1 Functional Requirements . . . 18
4.2.2 Non-functional Requirements . . . 18
4.3 Use Case Modeling . . . 20
4.3.1 Overview . . . 20
4.3.2 Detailed Use Cases . . . 21
4.4 Architecture . . . 23

5 Implementation **25**
5.1 Overview . . . 25
5.2 Surface Fire . . . 26
5.3 Fire Propagation . . . 28
5.3.1 Overview . . . 28
5.3.2 Burn Distances . . . 28
5.3.3 Minimal Time . . . 31
5.3.4 Iterative Minimal Time . . . 34
5.4 Fire Acceleration . . . 35
5.5 Crown Fire . . . 36
5.6 Spotting . . . 39
5.7 Simulation Progression . . . 43

6 Results **45**
6.1 Overview . . . 45
6.2 Spread Images . . . 45
6.3 Running Time Comparison . . . 47
6.3.1 Overview . . . 47
6.4 Kyle Canyon . . . 55

7 Conclusions and Future Work **58**
7.1 Conclusions . . . 58
7.2 Future Work . . . 58
7.2.1 Improved Spotting . . . 58
7.2.2 Atmospheric Incorporation . . . 59
7.2.3 Multi-GPU Implementation . . . 59
7.2.4 Visualization . . . 60

Bibliography **61**

List of Tables

4.1 vFireLib Wildfire Simulation Library Functional Requirements. . . . 19
4.2 vFireLib Wildfire Simulation Library Non-functional Requirements. . 19

6.1 The average time to transfer memory from the CPU host to the GPU device in the three largest simulation sizes. 50
6.2 A table containing the values for real-time versus simulation timesteps. Multiple simulation seconds are computed per real-world second. . . . 51

List of Figures

2.1 Neighbor access methodology for each of the propagation methods. From left to right: Minimal Time, Iterative Minimal Time, Burn Distances 6
2.2 Outline of the programming style between the CPU and GPU [23]. . 13

4.1 A use case diagram for the Wildfire simulation library. 20
4.2 A sequence diagram for the Wildfire simulation library. 21
4.3 A class diagram for the Propagation classes in the simulation library. 23
4.4 A class diagram for the Propagation classes in the simulation library. 24

5.1 Neighbor access methodology for each of the propagation methods. From left to right: Minimal Time, Iterative Minimal Time, Burn Distances 28
5.2 The possible error for a fixed time-step propagation method. (a) shows the initial step with two lit cells a and b propagating to cells a' and b'. (b) shows a' propagating to a" in the next time step. (c) is the situation where b" would have propagated faster to the slot occupied in the previous step by a", but because of the fixed time step, it would not propagate to an already lit cell. 30
5.3 An example of the problem syncing thread read access and write access. There is no way to stop one thread writing to another cell before the value is read by another cell. 33
5.4 A diagram to represent the phases of spotting, taken from [13]. 39

6.1 The Burn Distances Kernel burn pattern at a grid size of 512x512. . . 46
6.2 The Minimal Time Kernel burn pattern at a grid size of 512x512. . . 46
6.3 The Iterative Minimal Time Kernel burn pattern at a grid size of 512x512. 46
6.4 The runtimes to completion of simulation 48
6.5 The log-scale version of the graph found in Figure 6.4. 48
6.6 The log_2 scale of the throughput results. 49
6.7 The speedup between the parallel and sequential implementations. The average speedup was around 20X faster for all the versions. 50
6.8 The runtimes found for a single iteration of the kernel call with and without memory transfer time. 51
6.9 Effect of the wind on a kernel when the wind is a value of 40 in the positive x direction. 52

6.10 Effect of the wind on a kernel when the wind is a value of 40 in the positive x and y direction. 53
6.11 The prototype results for the spotting module. The wind in this figure is gusting at a value of 40 in the x direction. 54
6.12 The prototype results for the spotting module. The wind in this figure is gusting at a value of 40 in the x direction and 20 in the y direction. 54
6.13 The Kyle Canyon fire started at the entrance to the canyon after 5000 iterations in simulation. This simulation is timestamped at approximately 0.68 days or 16.4 hours. 55
6.14 The Kyle Canyon fire started at the entrance to the canyon after 7000 iterations in simulation. This simulation is timestamped at approximately 1.2 days. 56
6.15 The Kyle Canyon fire started at the entrance to the canyon after 10,000 iterations in simulation. This simulation is timestamped at approximately 2.1 days. 56

Chapter 1

Introduction

Every year, fighting forest fires costs taxpayers in the United States millions of dollars. From 2002 to 2012, the average amount spent per year on forest fire suppression by the federal government was $962 million, which only amounted to 32% of the entire federal wildfire protection funds [12]. This cost only covers the federal funds that are spent on fighting and preventing forest fires. It does not include the individual and environmental cost of loss of property and habitat. The highest cost that occurs during the efforts to fight a forest fire is the loss of life incurred by fire fighters. The ability to better predict the behavior of a wildfire greatly increases the effectiveness of fire fighting efforts, thereby reducing all the costs incurred during a forest fire. Another application domain for forest fires is the training of fire fighters. A sample forest fire could be provided in training scenarios which would allow fire fighters to have as close to hands-on forest fire training before they are exposed to real wildfires.

In order to simulate wildfires, scientists have developed several methods for modeling the propagation of fire [30, 2, 29]. These fire models are based on the properties of the environment in which the forest fire takes place. Such properties include, but are not limited to, fuel load, fuel type, wind, live moisture, dead moisture, and crown height. Incorporating these and other variables into spread models can allow for accurate prediction of where a fire will spread and how quickly it will arrive. Research into developing fuel and moisture models is active to this day. These models provide the basis for the properties on which forest fire simulation is based.

The ability to realistically simulate forest fires is desirable because it allows fire

experts to more accurately predict the impact of their fire-fighting decisions. Possible manipulations to the wildfire environment include adjusting moisture content to simulate a water drop, adjusting fuel loads where a simulated bulldozed treeline could exist, or reverse spread testing in which a fire started by firefighters would burn the fuel away from the advancing wildfire. Unfortunately, the amount of data required for realistic fire simulations requires a large amount of computation time to produce an accurate simulation. The more accurate and fine-grained the simulation, the longer it takes to process the data. A forest fire is a dynamic entity, therefore the ability of a simulator to run in real time is necessary for it to be an effective tool. The more complex and accurate a simulator is, the more useful it is to fire scientists. There are multiple aspects to accurately modeling the spread of a forest fire. The main four fire properties which influence the spread of a fire are base fires, crown fires, fire acceleration, and spotting [24].

Using the GPU as a general purpose computing device has become popular in recent years, especially on problems which require a large amount of data processing. The GPU is ideally suited to high volume data processing applications because it can processes millions of inputs simultaneously, while a CPU may only process up to a few at a time [31]. This work has developed a fire simulation library which allows a user to run base propagation with fire acceleration tests on real-world data, as well as testing the crowning conditions and a simple spotting method.

The remainder of this paper is structured as follows. Chapter 2 contains the background information on forest fire models and the existing forest fire simulators. Chapter 3 outlines of the work accomplished in this paper. Chapter 4 presents the main components of the library's software specifications and design. Chapter 5 describes the implementation details of the forest fire simulation library. Chapter 6 presents the results of the timing tests and fire simulations performed for this work. Finally, Chapter 7 presents ideas for future enhancements for the fire library.

Chapter 2

Background and Related Work

This chapter outlines the history of the work done in the research area of forest fire modeling and simulation. It then gives some background information on GPU computation as well as the fuel models used by this research.

2.1 Fire Models

2.1.1 Overview

The most widely used fire model is that developed by Richard C. Rothermel. His model of fire spread model depicts fire spreading in an elliptical shape. Rothermel also developed the first eleven fuel models that are still used to this day [30]. A fuel model is a model of a small region of forest and the vegetation it contains. Examples of vegetation types that are modeled by the fuel models are grass and grass-dominated regions, chaparral and shrub fields, timber litter, and stash [29]. The method by which the first eleven fuel models were created is the basis for the development of all modern fuel models. The fuel model contains information on properties of the forest in a particular region, and at a certain granularity. The forest is broken up into cells, each cell having properties which are modeled in the fuel model. For example, one fuel model might describe a coniferous forest in regions of 30x30 meter cells. These cells are what make up the basis for a fire simulation.

2.1.2 Taxonomy of Fire

There are three types of fire models: theoretical, empirical, and semi-empirical [24, 26]. Empirical models use statistical descriptions of wildfires based on controlled experiments in laboratories. However, their lack of incorporation with physical data limits their usefulness to controlled laboratory environments. Theoretical models are based purely in physical principles and used only for research purposes. Semi-empirical models incorporate some of the statistical modeling found in emperical models but also include experimentally derived approximations to portions of the models. Rothermel's spread equations are an important example of the semi-empirical fire models [30]. More detail on Rothermel's equations will follow in subsequent sections. The final type of models are theoretical, which rely solely on physical principles. Their limitation occurs at the boundary of what data is available. This work implements semi-emperical methods for calculating fire spread.

2.1.3 Fire Shape

The majority of the existing forest fire simulators, including this work, calculate wildfire spread based on the Rothermel's fire spread equations [30]. More detail on the simulators which use this fire spread model will be covered later in the section. Equation 2.1 shows his rate of spread equation, which is based on several parameters.

$$R = \frac{(I_p)_o(1 + \phi_w + \phi_s)}{\rho_b \varepsilon Q_{ig}} \tag{2.1}$$

Where R is the rate of spread(distance/time), $(I_p)_o$ is the no-wind propagating flux, ϕ_w and ϕ_s are the additional dimensionless propagating flux introduced by wind and slope respectively. The product of ρ_b and ε is referred to as the effective bulk density (heat/particle size). The effective bulk density models the amount of fuel per unit volume of the fuel bed raised to ignition ahead of the advancing fire. Q_ig is the heat of preignition (kiloJoules/kilogram). The heat of preignition is the heat required to bring a unit weight of fuel to ignition. These values are derived from or contained

in the fuel model that describes the cell for which the computations are being done.

The desired output from a forest fire simulator is a time of arrival map. Each cell in this map represents a cell in the simulation forest, and the value it contains is the time at which the cell ignited and started propagating the fire. Once a cell is lit, it begins contributing to the spread of the fire to the surrounding cells and continues burning until the fuel in that cell is entirely used. The method of propagation may vary between simulators, but the basic spread rate is usually based on Rothermel's equations.

2.1.4 Surface Fire

There are several potential approaches to calculating the propagation of the fire. This propagation also determines the method for stepping through time in a simulation. This work implemented three methods for iterating through a simulation to calculate the time of arrival map. The first two spread methods (Minimal Time and Iterative Minimal Time) are based on stepping through time independent of specific fuel conditions and are based on the paper by Sousa, dos Reis, and Pereira [35]. The third spread method implemented in this paper (Burn Distances) was based on code and methods found in vFire [16].

vFire implemented an accurate spread rate calculator based on Rothermel's fire spread equations and the fire spread and fuel model data to propagate based upon the physical burning of fuel. The Burn Distances propagation method was based on their work.

Sousa, dos Reis, and Pereira also used the GPU to improve their running times and ported fireLib to the GPU [35], but were the first to use the parallel programming language CUDA [23]. They implemented three kernels in which they explored three different propagation types. The propagation methods labeled Minimal Time and Iterative Minimal time are based on their work.

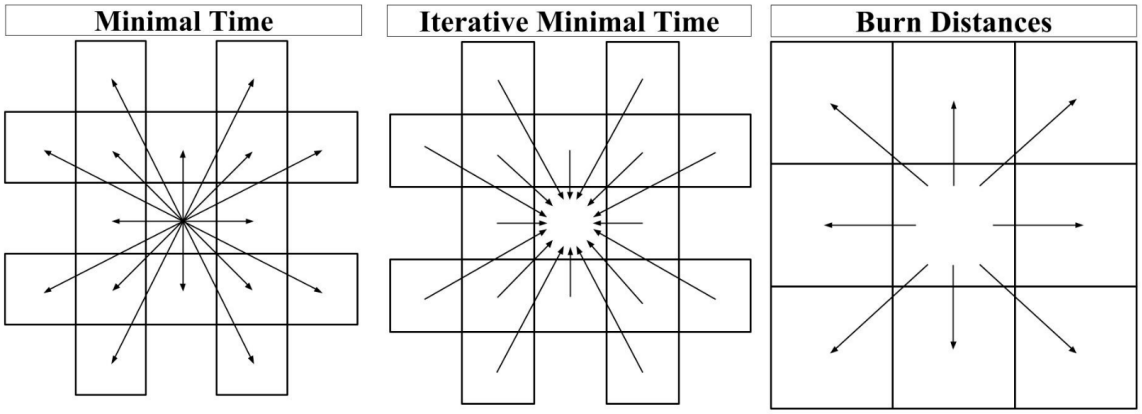

Figure 2.1: Neighbor access methodology for each of the propagation methods. From left to right: Minimal Time, Iterative Minimal Time, Burn Distances

Burn Distances The Burn Distances (BD) method is based on the idea that it takes a certain amount of time for fire to burn the distance between two cells. A distance is set as equivalent between all cells before the simulation burning may progress. The distance between cells is based on the properties of the forest model which is based on the size of the forest cell. The simulation iterates at a constant time step, and the amount the distance has burned is tracked throughout all the time steps. In this spread method, a cell looks to see which of its neighbors are on fire and which of those neighbors will burn the distance between the two cells first. The first cell to burn the total distance ignites the center cell as seen in Figure 2.1. In this work, the BD propagation method is based on work by Roger Hoang's vFire [16, 17]. The paper by Sousa, dos Reis, and Pereira also implemented a similar method, but it was not the basis for this propagation method in this work [35].

Minimal Time The Minimal Time (MT) propagation method uses a dynamic time stepping method to step through the simulation. At each time step, a cell is examined to see if it is on fire. If the cell is burning, then the neighbors of the cell are examined as seen in Figure 2.1. If the neighbor is already on fire, it is ignored. If the neighbor is unlit, then the time of arrival for that cell is computed using the propagation equation found in Equation 5.1. In the Minimal Time method, time is incremented

dynamically. Each time a new ignition happens, the ignition time is compared to the current 'timeNext' variable. If the new ignition time is smaller (the fire arrives sooner) than the current timeNext, then it replaces the timeNext value. This methodology means that time is incremented based on which cell will ignite the earliest. In the paper by Sousa, dos Reis, and Pereira, this method achieved the slowest speedup of their three methods [35].

Iterative Minimal Time The Iterative Minimal Time (IMT) method is designed to avoid data dependencies. Each cell operates independent of its neighbors, calculating the ignition time based on the spread rates, and only finishing when the values between step k and $k+1$ converge. Each cell looks at the minimal time for each of its neighboring cells to burn towards it as shown in Figure 2.1. The value is known to converge after the difference between two time steps is less than some small threshold. The most appropriate value for this threshold can be determined through experimentation. In the work by Sousa, dos Reis, and Pereira, this method achieved the highest speedup of their three methods [35].

2.1.5 Crowning

Crowning is the phenomena of the fire moving from spreading along the base of the forest to up the trees to their crowns. There are two types of crowning: passive and active. A passive crown fire is one which does not spread to the overall propagation of the fire. Conversely, an active crown fire does contribute to the spread of the fire. The modeling method used for this implementation will be discussed in the Implementation section of this paper. The method for calculating crowning in this library was based on the work found in vFire [16]. vFire based their implementation on the work found in FARSITE [10], which implemented their methods based on the work done by Van Wagner [36, 37].

2.1.6 Spotting

Spotting occurs when firebrands from the wildfire are lifted into the air and fly ahead of the advancing flame wall to start new fires. There are three sequential components to spotting: generation, transport, and ignition. There are different models that describe the fire brand generation, and it depends on many factors such as degredation of wood due to pyrolisis and combustion. Research in this area consists of experiments that determine the behavior of burning wood, and generates data as to the size of potential brands [21]. Fire transport depends on many factors including flame structures, weather models, and the aerodynamics around the fire brand itself. There are several layers to the behavior of a fire brand before it reaches the open air: between the point of origin of the firebrand and the top of the flame, while the firebrand is between flame tip and buoyant plume, and when it is in the buoyant plume [32]. The transport of the fire can be on a scale that is small, medium, or large [19]. The ignition following the transport of the fire brand is dependent on the heat of the brand as it lands and the fuel properties it encounters [18]. The studies that produce emperical data for igntion tests the success of a firebrand igniting a fire that successfully spreads a certain distance [28]. The implementation found in this work does not go into great detail on the generation and ignition. The models have been extremely simplified to fit in the scope of this work. The generation is modeled simply at the point when Torching occurs. Torching is the event when a fire spreads from a base fire up into the tops of the trees as crowning occurs. Ignition is modeled as a simple user-defined probability. This method was implemented based on the work found in FARSITE [10]. FARSITE based their implementation on the model developed by Albini in 1979 [**?**]. The implementation found in this work had to be simplified somewhat because of constraints on the data available to the researchers. Details on the limitations found in spotting may be found in the Implmentation chapter of this paper.

Another paper which implemented spotting did not use the same empirically-based methodology as the method used by FARSITE and in this paper. The work

by Koo, Linn, Pagni, and Edminster implements a method for spotting based on the theoretical models known about physical phenomena surrounding wildfires [18]. They experiment with different sized and shaped fire brands to determine the impact they have on the spotting distance possible during a forest fire.

2.2 Fire Simulators

Since Rothermel's paper was published in 1972, several fire spread simulators have been developed. Nearly every major forest fire simulator has used his spread equations as the basis for their simulation.

2.2.1 BEHAVE

The first major fire spread simulator was developed in 1986 called BEHAVE[2]. BEHAVE used Rothermel's fire propagation methods [30]. BEHAVE had two main functions to the application. The first function allowed users to load in fuel models from Rothermel's paper, but also to develop and save new fuel models. The simulator then had the ability to integrate the newly developed fuel models in its simulations. The second function of the application would run a simulation and burn prediction on the desired fuel model. The usefulness of the BEHAVE simulator is that it gives a realistic viewpoint of how a fire would behave given a specific fuel model at a specific instance in time. The output of this simulator appeared in a table which represented the times of arrival for each cell in the simulation. There was no visualization method available for this simulator. The simulation was meant to be used as a training tool rather than a real-time tool to be used to fight wildfires. It is still used to this day by fire scientists who are not familiar with the programming requirements of the newer simulators that have been developed. It remains a useful tool for fuel model development.

2.2.2 fireLib

A decade later in 1996, BEHAVE was the basis for a new forest fire library that was developed using the programming language C called fireLib [4]. The code was based entirely on BEHAVE's simulator, but brought up to a then-modern platform. FireLib can run much faster than BEHAVE, and the output is given in time of arrival arrays rather than a table. Each [x,y] in the array corresponds to a cell in the fire simulation, and the time of arrival is the time at which the cell ignites, and can then begin to propagate the fire. The fire library is more flexible than BEHAVE and allows a user to design their own methods for propagating the fire. There are a few different methods which may be used, and will be addressed later in the paper. However, where BEHAVE was an entire application which had an interface component, fireLib is simply an open source forest fire library and both the visualization and interface development are left up to the user. fireLib incorporates the element of time that BEHAVE is missing, allowing researchers to look at a fluid time scale rather than a single instance of a fire.

2.2.3 FARSITE

In 2004, FARSITE was developed, which works as a full-scale forest fire simulator [10]. It has been continuously developed since 2004 and is currently still operational. It incorporates more features than simple fire spread, such as crown fires, surface fires, fire acceleration, and spotting. While it is one of the most advanced and accurate forest fire simulators, it is not very fast. The FARSITE simulator runs entirely sequentially, which it will be shown how slow a sequential implementation of these methods are later in this paper. Despite the lengthly amount of time required to compute its simulations, it is one of the most widely used forest fire simulators in existence today. A benefit of using FARSITE over other options is that they are able to incorporate advanced geospatial data.

2.2.4 vFire

This paper used much of the fire spread implemenation from a forest fire simulator called vFire [17, 16, 15]. vFire was based on hFire, and are both cellular based spread models. They run faster than FARSITE, but do not have the same level of precision [27]. vFire was the first forest fire simulator to utilize the parallel nature of the GPU to run calculations on the fire spread. At the time of its development, the only way to program on the GPU was to use the programming language OpenGL [33]. vFire used the shader language glsl to utilize the multicore abilities of the GPU. This implementation of the forest fire tied the visualization and the data processing together, so they could not be run independently of one another. However, the visualization was designed to be much more immersive than what had been available before [25]. vFire implemented a technique that has dynamic time stepping to burn distances between cells to determine an accurate time of arrival for the fire spread. The important feature that vFire accomplished was porting the computation of the fire spread to the GPU using OpenGL shaders [33]. This simulator did not implement a sequential version to compare results against, and so there is no data to support how much improvement in runtime it accomplished.

2.2.5 FIRETEC

Researchers at Los Alamos National Laboratory have created a fire simulator which is not based on the elliptical spread equations developed by Rothermel. Rather than using semi-empirical data to calculate their spread methods, they use theoretical models of chemical reactions and heat transfers to descipher where the fire is going to propagate [20]. This method for calculating spread is very computationally expensive, and is used mainly for research purposes at the moment.

2.3 GPU Computation

2.3.1 Overview

GPU's were originally designed as graphics accelerators supporting only very specific fixed-function pipelines. This meant that using them for high performance computation was not easy unless it could be integrated with some sort of visualization language. An example of such an integration may be seen in vFire [16]. In 2006, NVIDIA released CUDA, which was the world's first solution to general-purpose computing on the GPU [23]. Ever since its release, and even beforehand, the GPU was used for its high-processing capabilities. The architecture of the GPU allows for thousands of low-powered processors to run in parallel. This type of computation is ideal for situations where the same computation to hundreds of inputs. The pitfalls of GPU computation occur when data dependencies exist in the data. If one portion of the data must wait on another to finish, it limits the usefulness of the GPU for processing.

2.3.2 CUDA

CUDA is a parallel computing platform and computing model created by NVIDIA [23]. It harnesses the hundreds of cores provided by a GPU an allows programmable kernels to be written. A kernel is a small bit of code that is run by a thread on the GPU. In the world of GPU programming, there is host data and device data. Host data is data which is stored on the home machine and processed by the CPU. Device data is data which is copied to the GPU for processing [31]. The architecture of a GPU that can be accessed through CUDA begins at the grid level. A single thread is assigned a thread ID. Each thread ID is unique among its block, which contains a certain number of threads. Many current GPUs have blocks that can contain up to 1024 threads. There may be many blocks in a single grid of a GPU, and the blocks may be assigned. The total number of threads would then be the number of blocks multiplied by the number of threads per block [23]. An outline of the heterogeneous programming style may be seen in Figure 2.2.

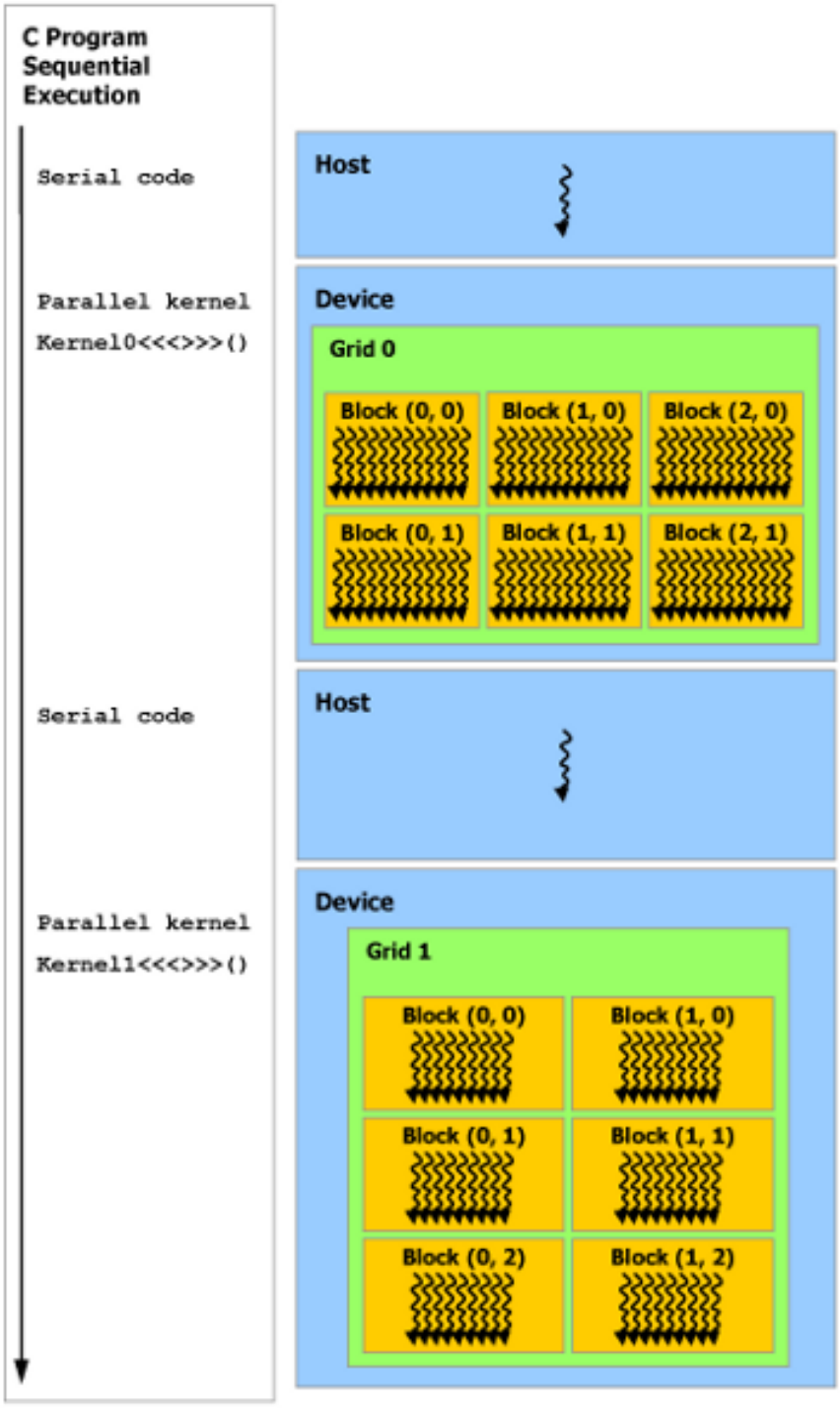

Figure 2.2: Outline of the programming style between the CPU and GPU [23].

In order to test the work presented in this paper, the initial tests were run on a CUBIX box [8]. The CUBIX box is designed to leverage multi-GPU and CPU programming. The box contains seven GPUs and four CPUs. This work only implements single-GPU simulations. The reason the CUBIX box was used was for the speed of determining results. It allows several simulations to be run and timed simultaneously, which meant the results could be gathered faster. However, due to technical challenges, the final results were timed without using the CUBIX machine.

2.4 Fuel Models

The data needed to process the fire simulation begins with the Fuel Models required by the simulator. Rothermel developed thirteen original fuel models, which are still used today [29]. Additional fuel models have been developed since that time, 42 of which are incorporated into this work. Additional fuel models, including those of what are called 'unburnable' in this work may be found in the work by Scott and Burgan [**?**]. The fuel models contain data regarding surface-area-to-volume ratio by class and component, fuel model type, fuelbed depth, extinction moisture content, and fuel particle heat content. These data types are used to determine the behavior of the fire in a particular cell of the simulation. The details on the data available may be found in the documentation of the library.

Chapter 3

Real-time GPU-based Wildfire Simulation

The proposed vFireLib is a wildfire simulation library that allows the user to take advantage of the highly-parallel nature of the GPU. The propagation, crowning, and spotting are all implemented both sequentially and as kernels on the GPU using the programming language CUDA [23]. The novelty of this work resides in the comprehensive implementation of a forest fire simulation library that can leverage the highly-parallel nature of the GPU.

Although several fire simulators have been implemented in the past, including BEHAVE, FireLib, and FARSITE, they do not use the GPU to compute the fire spreads [4, 2, 10]. Because of their sequential nature, as well as the high demands on amount of data needed for an accurate simulation, the time required to run these simulations make them unsuitable for real-time applications such as training or live wildfire prediction. The dynamic nature of a wildfire makes the need to run in real-time not only helpful, but necessary. In order to make a simulator operate in real-time, a tradeoff exists between accuracy and processing time. It is not cost-effective to expect fire fighters to have access to multi-CPU processing platforms, but the cost of a single GPU is not unreasonable to expect for a wildfire expert to obtain. For these reasons, and reasons of ease of use given the programming language CUDA, wildfire simulation on the GPU is the focus of this work. The only previous work to port the computation load of propagation simulation to the GPU was vFire [16],

which was implemented before programming for the GPU became easily accessible.

The nature of wildfire simulation begins with the need to step through time to give an accurate estimation of the behavior of the wildfire. At each time step, computation must be done on every cell of the fire. At each time step the spread of the fire, the acceleration of the fire, the crowning test, and the spotting check all need to be calculated. Leveraging the GPU to operate as the computational workhorse is ideal because every cell needs the same processing done on it, no matter what the environment. The GPU can process each cell in parallel to its neighbors.

This parallel computation allows each time step to be processed as one pass on the GPU. While the cores on the GPU are not as powerful as CPU cores, the real benefit to this computation style comes when many passes over the data are needed. Passing data between the GPU and CPU is a bottleneck as far as speed is concerned. This is why running all the computations on the GPU is ideal for the simulator. At a fine granularity, the benefits of speed in processing time outweigh the negatives of having to pass data back and forth between the CPU and GPU.

The focus of this work is improving the runtime needed for the simulation of a wildfire by using the GPU as the processing workhorse. The model developed is based upon the existing work found in many simulators including vFire and FARSITE. This work moves beyond the implementation of the work found in vFire, which can be thought of as the parent research to this implementation. Not only does this work reimplement the work of vFire on the modern platform for GPU computing by using CUDA, but adds additional functionality in its three propagation methods and the implementation of a basic spotting model.

The work presented in this paper will be compiled as a library to easily be expanded into a visualization platform. The library will have all the functionality needed to perform fire simulation without any complex visualization tools as well. The use of the library and its tools is left up to the user. The granularity of simulation can be adjusted to incorporate more detail as needed to the simulation. Stepping through time is determined by which spread methodology is chosen for the simulation. Timings

can be performed on the granularity of the results to determine if there is an optimal granularity for real-time fire performance.

Chapter 4

Specification and Design

4.1 Overview

The vFireLib library provides functions that allow for a dynamic forest fire simulation to occur. The library is responsible for managing the data provided by input files by a user, calculating spread data, running the sequential or parallel simulation, and writing out the final data to a file for further reading. The diagrams in this section of the paper were generated using the Creately web interface for generating UML diagrams [9].

4.2 Service Requirements

4.2.1 Functional Requirements

The functional requirements were created from the behavioral requirements of the library's functionality as used by an outside source. The list of functional requirements may be viewed in Table 4.1. The software engineering steps used in this work were based on the work accomplished by Ian Sommerville in his Software Engineering book [34].

4.2.2 Non-functional Requirements

The non-functional requirements were created based on the internal interactions of the library functions. Data needed to be manipulated and passed among classes and between the host and device for computation purposes. The non-functional

Table 4.1: vFireLib Wildfire Simulation Library Functional Requirements.

Number	Description
FR01	The library shall read in fuel model, moisture model, and terrain data.
FR02	The library shall define the granularity of simulation size.
FR03	The library shall create and initialize all data members necessary to a complex simulation on the CPU and GPU.
FR04	The simulation shall calculate the maximum rate of spread for every cell.
FR05	The library shall run a GPU simulation using the BD, MT, or IMT spread methodologies.
FR06	The library shall calculate fire acceleration for the simulation.
FR07	The library shall allow the Crowning flag to be toggled.
FR08	The library shall allow the Spotting flag to be toggled.
FR09	The library notify the user when their input files are not compatible with the simulation system.
FR010	The data must be initialized before the simulation is run.

Table 4.2: vFireLib Wildfire Simulation Library Non-functional Requirements.

Number	Description
NR01	The library shall be written in C++ and CUDA.
NR02	The library must use CUDA version 6.0 or higher.
NR03	The library's sequential implementation must be a direct reflection of its parallel implementation.
NR04	To run the parallel version of the code, an NVIDIA GPU with CUDA Compute Capability 2.0 or higher is needed.
NR05	This library must be used on the Linux platform.
NR06	The library must use GDAL version 1.10.1.
NR07	The library must be compiled with CMake Version 2.8 or higher.

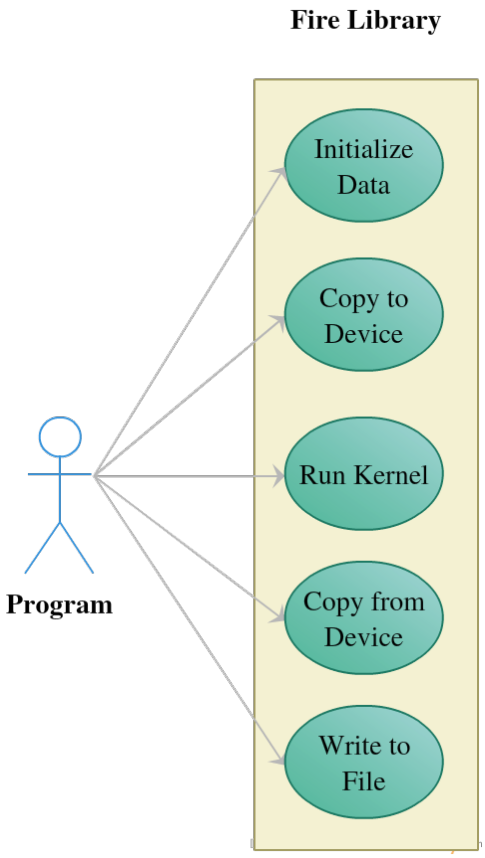

Figure 4.1: A use case diagram for the Wildfire simulation library.

requirements are listed in Table 4.2.

4.3 Use Case Modeling

4.3.1 Overview

For a better understanding of how the user interacts with the library, this section describes use cases for running a simulation. These diagrams have been generated using the Unified Modeling Language [3]. Both the parallel and sequential functionalities will be shown here. The use case model may be found in Figure 4.1. In order to better illustrate the library's functionality, a sequence diagram may be seen in Figure 4.2.

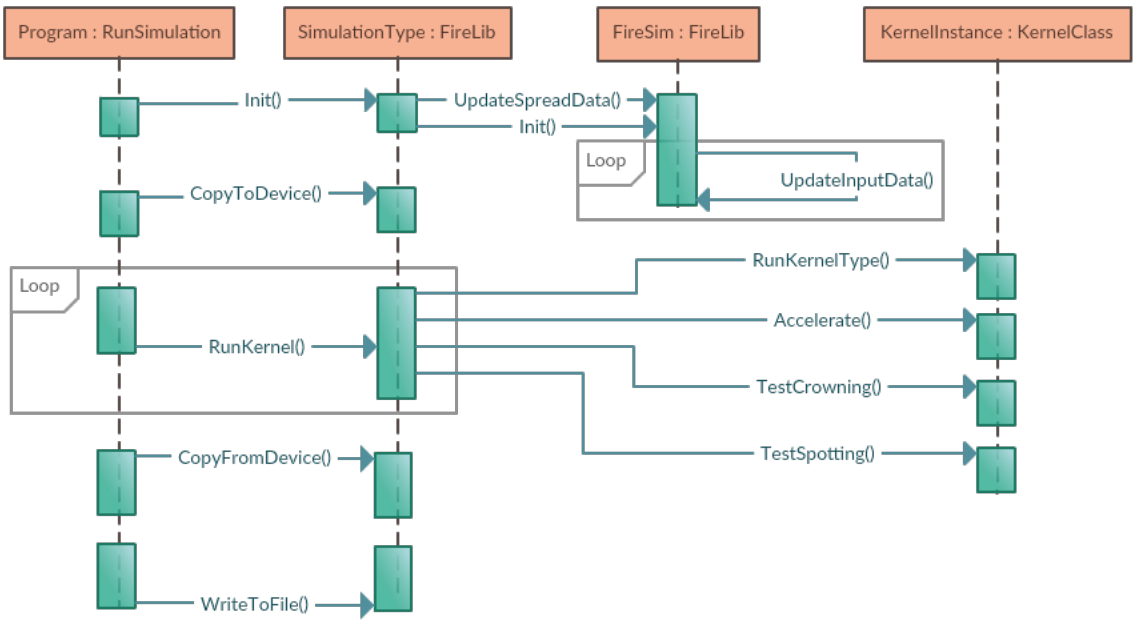

Figure 4.2: A sequence diagram for the Wildfire simulation library.

4.3.2 Detailed Use Cases

4.3.2.1 Initialize Simulation Data

The terrain data encompasses more than the terrain data. The fuel model data must be read from the files, the terrain data, the moisture data, the canopy height data, the crown base height data, and the crown bulk density data. If there is an error with any of these files, the library reports an error. This is the point where the simulation is scaled to the appropriate granularity.

4.3.2.2 Initialize CPU Data

The data must be initialized from the input data into a format which is useable by the simulation. This includes calculating the maximum spread rate before the simulation begins. This function must be called whenever there is a change to the data values in the initialize function.

4.3.2.3 Copy To Device

The data needed for the simulation must be copied from the host to the device (GPU). This must occur every time the data is changed on the CPU side of the library.

4.3.2.4 Propagate

This occurs when a propagate function is called. This could be from three sources: BD, MT, or IMT.

4.3.2.5 Accelerate

Once the fire has spread after a time tick, the rate of spread from the fire is accelerated.

4.3.2.6 Crowning

Once the fire has been accelerated, a test is done to see if the fire is crowning. The moment a fire crowns, the torching flag is set for one time tick. The crowning test also checks to see if the fire is passive. If it has progressed to an active fire, the maximum rate of spread is updated to reflect the change.

4.3.2.7 Spotting

In the event that torching has occurred during the crowning phase, spotting is calculated. A fire brand is launched into the air and it is tested to calculate if any new fires start ahead of the flame wall.

4.3.2.8 Copy From Device

On completion of the desired simulation, the data from the simulation must be copied back from the device back to the host for further processing.

4.3.2.9 Write To File

This functionality allows the user to write the computed output of their time of arrival map to a .csv output file.

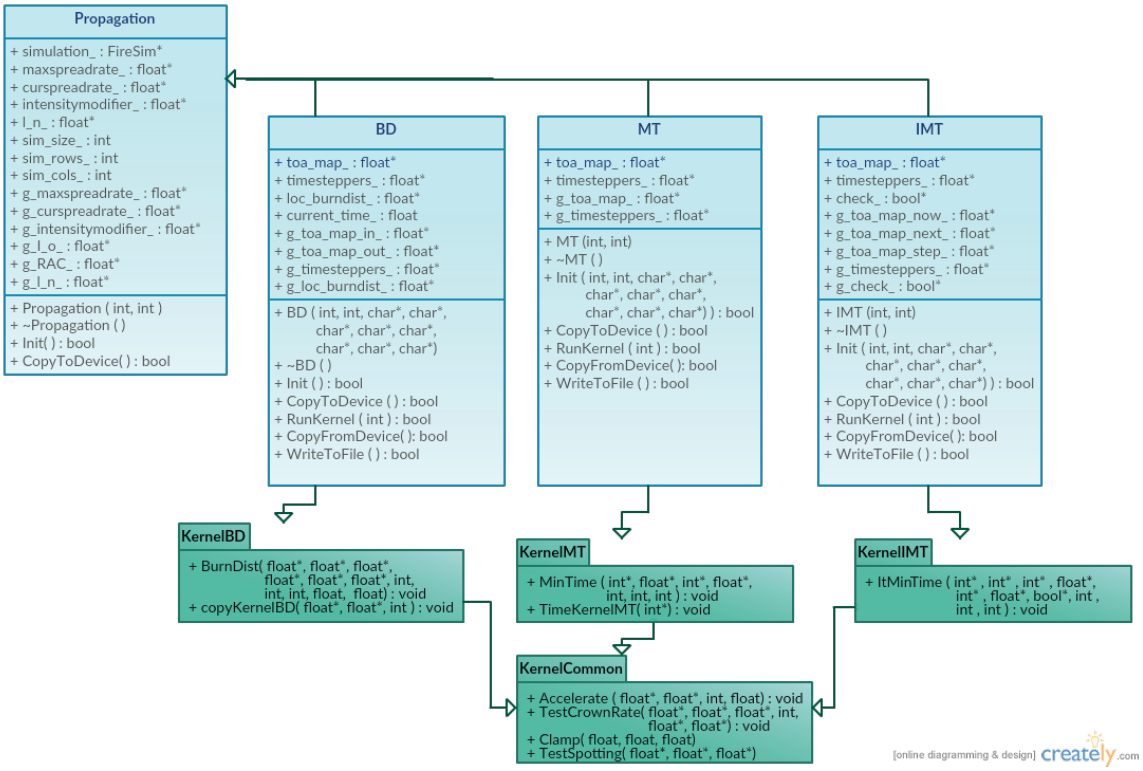

Figure 4.3: A class diagram for the Propagation classes in the simulation library.

4.4 Architecture

This library is designed to act as a facilitation tool that will allow a user to program their own custom-defined simulations. The program aims to make visualization easily portable to users. In order to convey an understanding of the overall architecture of the library, a class diagram is provided to describe the system in Figures 4.3 and 4.4:

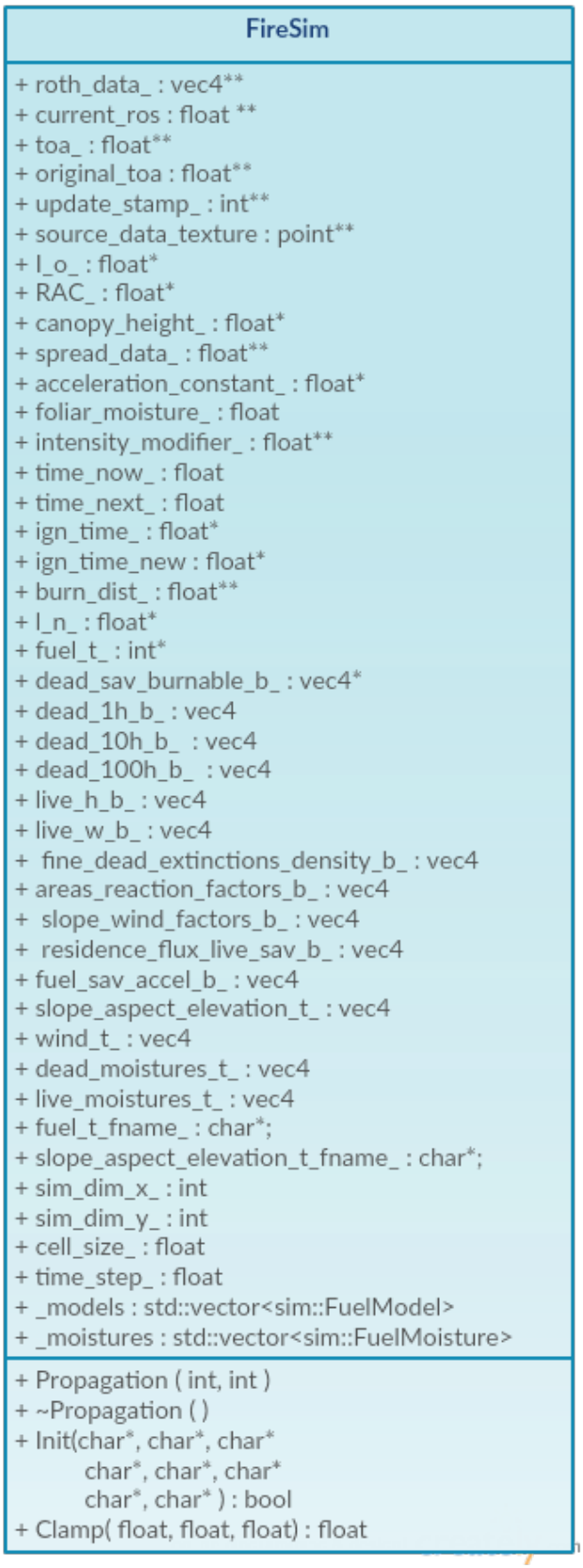

Figure 4.4: A class diagram for the Propagation classes in the simulation library.

Chapter 5

Implementation

5.1 Overview

Once the preprocessing is complete, one of three simulation methods was applied: Minimal Time, Iterative Minimal Time or Burn Distances. Since the goal of a forest fire simulator is to operate in real-time, it is unrealistic to expect to wait that long to receive the simulation data. After thirty minutes, the fire will have changed enough for the simulation's predictions to be irrelevant. The following algorithm outlines the simulation steps from start to finish:

Algorithm 5.1 Simulation Progression

```
InitializeTerrainData();
CalculateSpreadRates();
while Simulation !Complete do
  RunPropagationSimulation();
end while
GenerateOutputFile();
```

The InitializeTerrainData() and CalculateSpreadRates() functions in Algorithm 5.1 are part of the preprocessing in this project. These are not addressed in detail in this paper due to space constraints. The RunPropagationSimulation() portion is the focus of this paper and will be outlined in more detail in the following sections.

In order to address the runtime analysis needed for this paper, a sequential version of the kernels was also implemented. The equations and spread methodologies are the same between the kernels and the sequential implementation. However, many

of the data concurrency problems were not a problem in the sequential version, and therefore some of the secondary kernels which will be addressed were not implemented in the sequential versions. The sequential implementations are not discussed in detail in this paper because they are not the main focus of the library and exist mostly for comparison purposes.

5.2 Surface Fire

There are several potential approaches to calculating the propagation of a wildfire. This paper implemented three methods for iterating through a simulation to calculate the time of arrival map for a simple one-source forest fire under constant terrain and wind conditions. The first two spread methods (Minimal Time and Iterative Minimal Time) are based on stepping through time independent of specific fuel conditions and are based on the paper by Sousa, dos Reis, and Pereira [35]. The third spread method implemented in this paper (Burn Distances) was based on code and methods found in vFire [16]. vFire implemented an accurate spread rate calculator based on Rothermel's fire spread equations and the fire spread and fuel model data to propagate based upon the physical burning of fuel.

The model for propagation rate used in this paper was the same for all three spread methods. The model is based on Rothermel's fire spread equations, and is found increately Equation 5.1. This equation was derived by the creators of vFire [16], and the data processing done in the preprocessing phase of this project was based on their work. The preprocessing step translates the equation from Rothermel's work seen in Equation 2.1 to what is seen in the following Equation 5.1. The preprocessing data is out of the scope of this paper and is not covered in detail.

$$r(\Theta) = R_{max}\frac{1.0 - \varepsilon}{1.0 - \varepsilon cos(\phi - \Theta)} \tag{5.1}$$

R_{max} is the maximum rate at which a fire can spread. The scale at which it spreads is dependent on the input files. It will be in distance / time step size. In

this implementation, this rate will be in meters/100 seconds. ε is the eccentricity of the fire, which is based on wind and slope data. ϕ is the orientation. Θ is the direction in which the fire is spreading. R_{max}, ε, and ϕ are all computed based on the terrain data before the propagation simulation takes place. This is done in the preprocessing stage because the rates at each cell do not change until the forest model changes. An interactive simulator could potentially allow for these variables to change (i.e. modeling a water dump from a helicopter or a bulldozer tearing down a line of trees) but that is outside the scope of this research paper. To implement these features, changes would be made to the fuel and moisture models used to determine the possible rate of change in a cell. The direction Θ is computed based on which neighbor is being examined at the time.

During the preprocessing phase, there are several data files which need to be processed and interpolated to be of the same size. The fuel data and slope data are stored in files containing interpolation data such as size of cell, width, and height of the data grid. The Geospatial Data Abstraction Library (GDAL) was used to interpolate the data from the terrain and fuel files into the desired size of simulation [11]. Wind data is incorporated into the spread rate calculations as a 2D vector for each cell in the grid. The wind data contains a direction and magnitude. The fuel models provide the detailed parameters by which the rate of of spread is calculated. There are potential areas in this processing phase (such as calculating the Rothermel spread properties) that could be parallelized to improve overall running times of this simulator. However, the focus of this paper was to explore the potential for calculating fire spread on the GPU, so these possibilities were not addressed and are left for future work.

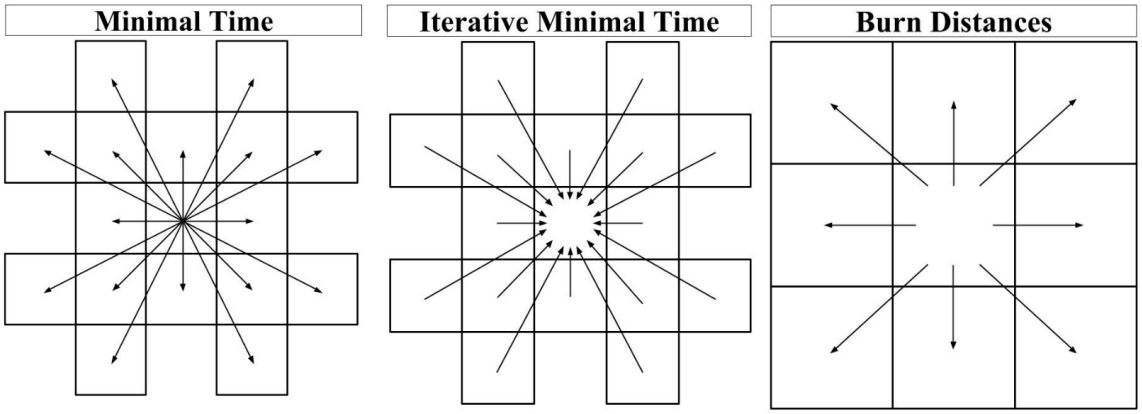

Figure 5.1: Neighbor access methodology for each of the propagation methods. From left to right: Minimal Time, Iterative Minimal Time, Burn Distances

5.3 Fire Propagation

5.3.1 Overview

In a wildfire spread simulation, there has to be a way to iterate through time and check for the propagation of the fire. This thesis implemented three such spread methods, which provide different fireline shapes as they move through the simulation. Each of the propagation methods are written as CUDA kernels [23]. The implementation details of each unique method follow in this section of the paper. The following figure, Figure , is a repeat of Figure 2.1 for ease of reference to the reader. It depicts the neighbor access methods for the propagation methods.

5.3.2 Burn Distances

Burn distances is a kernel which simulates the burning of the physical distance between fire cells. In order to limit unnecessary computations, cells that are already on fire are skipped in the kernel. Only those that are not on fire check their neighbors to see if during this time step they are set alight.The pseudocode for the algorithm for the BD propagation method is found in Algorithm 5.2.

Algorithm 5.2 Burn Distances Algorithm

```
for cell = 0 to numCells do
  // Check to see not on fire
  if ignTime[cell] == INF then
    Skip
  end if
  // Check Neighbors for spread
  for n = 0 to 7 do
    if ignTime[neighborCell] < INF then
      Skip
    end if
    ROS = Compute ROS according to Equation 5.1
    burnDistance(totDist[neighborCell], ROS, timeStep)
    if distance is burnt then
      ignTime[neighborCell] = timeNow
    end if
  end for
end for
```

During each kernel call, each cell is processed independent of its neighbors. However, when a new cell ignites, the data must be written out to the vector somehow. There is no way to ensure which threads are reading and writing all at the same time. In fact, it was found that race conditions existed when the threads were reading from and writing to the same vector. In order to fix this problem, an input and output vector were created. Threads read only from the input vector and write to the output vector. A secondary kernel was written to copy the data values from one vector to another between the propagation kernel calls. The algorithm for the copy kernel may be found in Algorithm 5.3:

Algorithm 5.3 Burn Distances Algorithm

```
for cell = 0 to numCells do
  // Copy from output back to input
  InputVector[cell] = OutputVector[cell]
end for
```

An issue with this method arose from the static time step and needed to be addressed. Figure 5.2 shows the problem that can arise from the fixed time step. Figure

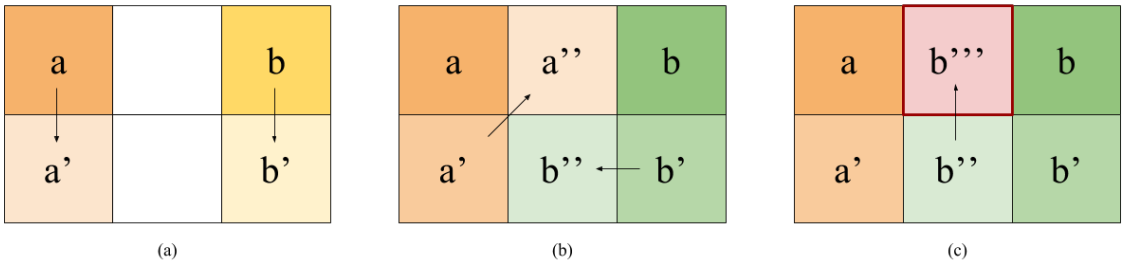

Figure 5.2: The possible error for a fixed time-step propagation method. (a) shows the initial step with two lit cells a and b propagating to cells a' and b'. (b) shows a' propagating to a" in the next time step. (c) is the situation where b" would have propagated faster to the slot occupied in the previous step by a", but because of the fixed time step, it would not propagate to an already lit cell.

5.2 (a) shows the initial propagation. In this scenario, imagine that the propagation rate for b is much higher than a, but because the time step is large enough, they both propagate approximately one cell per time step. Figure 5.2 shows the second propagation step in time. In this example, a' propagates to a". Because of the faster propagation rate of b, b' should propagate to b" and b" to b"' before a' propagates to a". In the case where the time step is too large, the time of arrival would show an erroneous value for the cell holding a". To fix this issue, the time step in the simulation must be set to a small enough value to avoid this error. The time step should be set to the smallest possible time it takes fire to propagate from one cell to another.

A cell ignites when the distance from an ignited neighbor has been completely burned. Each cell stores the distance that has been consumed by the fire in each direction to its neighbors. The simulation checks a cell's neighbors for ignition, and then uses their properties to burn the amount of distance in that time step towards the current cell as seen in Figure 5.3.1. The cell ignites when one of its neighbors burns the distance completely. The cell then inherits the properties of the fire that propagated to it. The equation to determine how much distance is burned can be

found in Equation 5.2.

$$d = d - r\Delta t \tag{5.2}$$

Where d is the distance that needs to be consumed. It is slowly decremented over time by the rate of spread (r) times the time step size ($\Delta\ t$). In order to account for the fact that an 'overburn' could occur with fixed time steps. The exact time of arrival that is calculated is dependent on the exact time of arrival that the fire would have arrived at the cell. The equation used to find the exact time of arrival is found in Equation 5.3.

$$TOA = t_{now} + \frac{d_{over}}{r} \tag{5.3}$$

Where TOA is the time of arrival that is written out to the output time of arrival map, t is the time in the simulation during which the propagation is occurring, d_{over} is the distance the fire burnt past the desired difference, and r is again the rate of spread.

5.3.3 Minimal Time

The Minimal Time algorithm was developed using the information provided in Sousa, dos Reis, and Pereira's paper [35]. In this propagation method, each cell spreads outwards towards its neighbors. At each time step, a cell is examined to see if it is on fire. If the cell is burning, then it calculates the propagation time to the surrounding cells. If any of the neighbors are on fire, it is ignored. If the neighbor is unlit, then the time of arrival for that neighbor is calculated. A neighbor which has been lit in the same timestep as the current spread calculations will still be examined for the possibility of a sooner time of arrival from the current cell. A difference between this kernel and the Burn Distances kernel is the time stepping is dynamic. Each time a cell is successfully ignited, the new time of arrival is compared against a 'timeNext'

variable to see if it a sooner time of arrival. The next step in time is based on the time in the simulation when the soonest cell ignites. The algorithm for the Minimal Time kernel may be found in Algorithm 5.4.

Algorithm 5.4 Minimal Time Algorithm

```
for cell = 0 to numCells do
  if timeNext > ignTime[cell] AND ignTime[cell] > timeNow then
    timeNext = ignTime[cell]
  else if ignTime[cell] == timeNow then
    // Propagate Fire
    for n = 0 to 15 do
      //If neighbor is unburned
      if ignTime[neighborCell] > timeNow then
        ROS = Compute ROS according to Equation 5.1
        ignTimeNew = timeNow + L_n/ROS
        if ignTimeNew < ignTime[neighborCell] then
          igntime[neighborCell] = ignTimeNew
        end if
        if ignTimeNew < timeNext then
          timeNext = ignTimeNew
        end if
      end if
    end for
  end if
end for
```

There are a few challenges which present themselves when implementing the algorithm. As in the Burn Distances kernel, data synchronization is an issue. Each cell in the fire is processed by one thread at a time, but every thread need access to the time of arrival map for both reading and writing. However, the solution to the problem between the two kernels is different. In order to minimize these race conditions, atomic operations were used to ensure that data integrity is maintained. CUDA's AtomicMin() operation was used to ensure that a cell that was writing to a data location was not overwriting a smaller time of arrival, which is the correct value that needed to be stored [23]. Atomic operations in CUDA are designed to lock read-write access so a single thread has exclusive access rights. The AtomicMin() operation ensures that the minimum of two integers is the one which is stored at

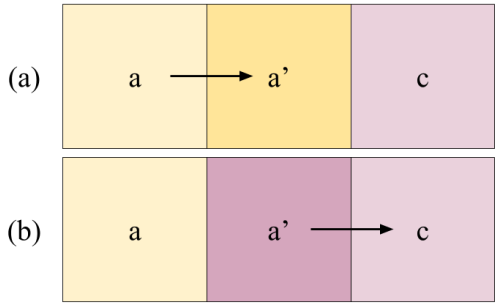

Figure 5.3: An example of the problem syncing thread read access and write access. There is no way to stop one thread writing to another cell before the value is read by another cell.

the memory location [31]. This solves the problem of one thread reading a value for its comparison then another writing a value to that same location, which would mean the value against which the calculations are compared would be inaccurate. A visual example of this read-write problem can be seen in Figure 5.3. The reason the AtomicMin() function is a solution to the read-write problem for this kernel as opposed to the Burn Distances kernel is that the Minimal Time Kernel processes integers. AtomicMin() has no implementation for floating point precision data values.

A secondary kernel was introduced to manage the time update between time steps in the simulation. The secondary kernel was found to be the most efficient solution to this problem. CUDA does not allocate threads in any specific order, which means thread 1 could be the last to finish calculations while thread 1,000 could be the first [31]. In order to step through time in the MT propagation method, the timeNext variable needs to be updated after all calculations finish. Since there is no way to ensure which thread would be the last to finish operating on the data, another method for iterating the variable was needed after all threads had finished their computation. The secondary kernel is called after the first terminates, which ensures that all threads have finished their computation before the timeNext variable is updated. Copying the two-element time vector back to the host device and managing the update there was also tested, but it was found to be faster to write a new kernel to update the data without copying anything back to the host device.

5.3.4 Iterative Minimal Time

The IMT kernel is very similar in nature to the MT kernel, but accesses neighbors in the opposite manner. The same data synchronization issues are also found in this kernel, where reading from and writing to the same cells would cause race conditions. A simple example of this problem may be seen in Figure 5.3. There is no way to stop a thread from writing to an output position before another thread reads in the data it needs to do its own spread calculations. Figure 5.3 (a) shows the writing of the value from a to a'. The value that existed before a' was the appropriate value for c to read in to do its calculations. Instead as seen in Figure 5.3 (b), it is the result value from a that it reads into do its calculations. This error causes race conditions to occur and artifacts to appear in the simulation. Instead of using atomic operations to solve this problem, two time of arrival vectors are passed to the kernel at startup. The two kernels act as an input and output kernel, reading from the former and writing to the latter. This introduces a new problem with maintaining accurate spread data across the input and output vectors. The secondary kernel in the IMT method is both used to copy data from the output back into the input as well as checking for the terminating condition for the simulation. The algorithm for the IMT kernel may be found in Algorithm 5.5.

Algorithm 5.5 Iterative Minimal Time Algorithm

```
for cell = 0 to numCells do
  // Check for simulation completion:
  if |ignTime[cell] − ignTimeNext[cell]| < thresh then
    //Mark as converged
  end if
  if ignTime[cell] > 0 then
    ignTimeMin = INF
    //Propagate Fire
    for n = 0 to 15 do
      ROS = Compute ROS according to Equation 5.1
      ignTimeNew = timeNow + L_n/ROS
      ignTimeMin = MIN(ignTimeNew, ignTimeMin)
    end for
  end if
end for
```

5.4 Fire Acceleration

A fire does not begin burning at its maximum rate of spread. Instead, it accelerates slowly towards its maximum rate over time as it burns. In this simulator library, fire acceleration is based on the work done by vFire [16]. vFire in turn used the model presented by FARSITE [10]. The formula is a simple logarithmic model that state the rate at some time is dependent on the time allowed for the fire to accelerate to its maximum rate depending on the current conditions of the cell in which it burns. This model was developed by the Canadian Forestry Fire Danger Group and Mcalpine and Wakimoto [14, 22].

Given the maximum spread rate R_{max}, the maximum spread rate at time t is given by:

$$R(t) = R_{max}(1.0 - e^{a_a t}) \tag{5.4}$$

where a_a is the acceleration constant and is drawn from the fuel model appropriate for the current fire cell. The time t_{max} required to achieve the maximum spread

rate given the current spread rate R is given by:

$$t_{max} = \frac{1.0 - \frac{R}{R_{max}}}{a_a} \tag{5.5}$$

To calculate how much the rate should change per time step, the spread rate for every cell is increased at each time step. The rate of increase dR is given by:

$$dR = \frac{dt}{t_{max}}(R_{max} - R_{current}) \tag{5.6}$$

A newly ignited fire will begin with a spread rate of zero and increase steadily to its maximum rate. Once the maximum spread rate is reached, the simulation does not accelerate past it. As the fire spreads to a new cell, that cell inherits the rate properties of the cell which ignited it. If the maximum spread rate of the newly ignited cell is smaller than the rate it inherits, it is clamped to its maximum. If the rate is slower than the maximum spread rate of the newly ignited cell, it is accelerated using the following equation:

$$dt = timeOfArrival(thisCell) - max(baseTime, timeOfArrival(ignitingCell)) \tag{5.7}$$

The time of arrival issue referred to by Figure 5.2 was a problem caused by acceleration in vFire. However, the time stepping methods in the MT and IMT methods do not have this issue as they have dynamic timestepping, and so they are not bound by the same limitations. The BD method's fix was addressed earlier in the chapter.

5.5 Crown Fire

The occurrence of crown fires is important for both the overall spread of the fire as well as the phenomena of spotting. Torching occurs when the crown fire spreads from

the base of the trees to the top of the trees, and is the moment when the possibility of spotting occurs. The crown fire begins as passive and has the ability to move to an active state, which increases the maximum rate of spread by the fire. The implementation presented in this paper is again based on the implementation found in vFire [16]. Their method was based entirely on FARSITE's implementation, which uses Van Wagner's methods found in [36, 37].

In this implemenation, the maximum passive crown rate is given by:

$$R_{max_{crown}} = 3.34R_{10} \tag{5.8}$$

The necessary components needed for calculating the actual maximum spread rate of an active crown fire in addition to the base propagation are all calculated based on properties found in the fuel model for each cell. The threshold which determines the point at which a passive crown fire is promoted to active (RAC) is given by:

$$RAC = \frac{3.0}{CBD} \tag{5.9}$$

The reaction intensity (I_b) of the fire can be obtained by multiplying the current spread rate by the intensity modifier. The intensity modifier is calculated using the same data which goes into the Rothermel equations.

$$I_b = R * IntensityModifier \tag{5.10}$$

I_o is the threshold intensity for a crown fire to occur. The moment when this threshold is passed is the moment when Torching occurs and a flag must be set for the spotting check. I_o is given as follows:

$$I_o = (0.010CBH(460 + 25.9M)) \tag{5.11}$$

Where CBH is the crown base height and M is the foliar moisture content. The crown base height is stored in one of the input files needed to operate this simulator. In this implementation (as well as the one found in vFire), foliar moisture is simply set to be 1.0. In the context of the simulation, the I_o values only need to be calculated once and are included in the preprocessing phase of the simulator.

In order to determine the actual spread rate of the fire after crowning, there are two parameters which must be calculated: the crown coefficient (a_o), and the surface fuel consumption (R_o). R_o can be calculated by using the reaction intensity of the fire, the current rate of spread, and the threshold intensity, given by:

$$R_o = I_o \frac{R}{I_b} \tag{5.12}$$

The crown coefficient a_c may then be calculated as follows:

$$a_c = \frac{-ln(0.1)}{0.9(RAC - R_o)} \tag{5.13}$$

The crown fraction burned (CFB) of the cell determines the actual maximum current spread rate of the crown fire. The CFB is found using the following equation:

$$CFB = 1 - e^{-a_c(R-R_o)} \tag{5.14}$$

In order to determine the actual maximum rate of spread, the current rate is added to the CFB that is scaled based on how close to the maximum crown rate the current rate is, as given by:

$$R_{actual} = R + CFB * (R_{max_{crown}} - R) \tag{5.15}$$

The maximum between this new calculated value and the maximum spread rate for the base fire is the actual value that is passed back as the maximum spread rate of a fire cell containing an active crown fire.

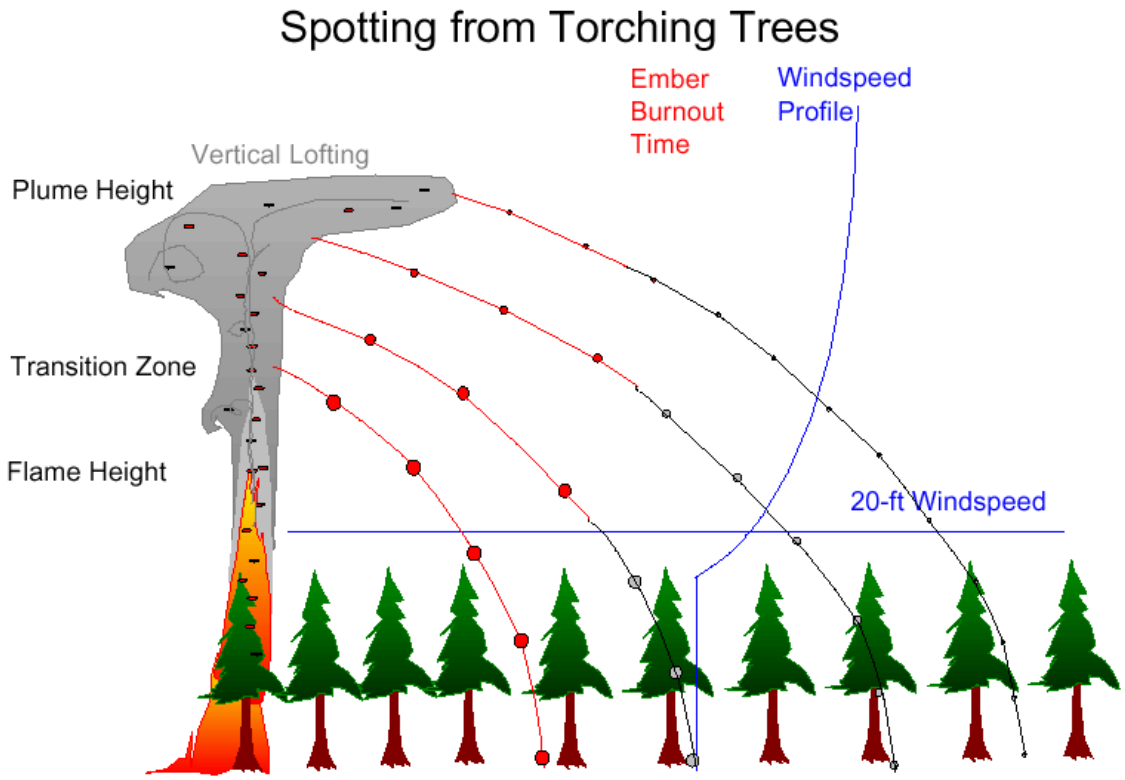

Figure 5.4: A diagram to represent the phases of spotting, taken from [13].

5.6 Spotting

Spotting is the phenomena that occurs when a fire brand is lofted into the air and blown ahead of the fire to potentially start a new fire ahead of the advancing fire wall. A fire brand is a burning piece of wood from a tree or debris found in the burning fire cell [24]. The implementation found in this paper is based on what was developed in the FARSITE fire simulator [10]. FARSITE uses Albini's 1979 model for spotting propagation to determine the location of the potential new fire [1]. In this implementation, the fire brands are assumed to be cylinders, which is an approximation to simplify the aerodynamics of the model. Figure 5.4 shows the stages of spotting and will be referenced for the rest of this section as the implementation of spotting is described.

When torching occurs, the spotting calculation can begin. The first important

component which needs to be computed is the time it takes for the particle to reach its maximum height t_t. t_t is dependent on four time components given by:

$$t_t = t_0 + t_1 + t_2 + t_3 \tag{5.16}$$

Where t_0 is the time of steady burning tree crowns. This is a data value that is dependent on the properties of the forest in the cell where spotting is occurring. This value was not available for this work, so a placeholder value was created for the time being and is a constant value for the simulations in the results section of this paper.

t_1 is the time it takes a particle to travel from its point of ignition to the flame tip. This paper makes the assumption that the fire brands originate from the top of the tree crown, but the actual value could be anywhere between the first branches on the tree capable of producing a brand to the tree crown. The equation used to calculate t_1 is as follows:

$$t_1 = 1 - (\frac{z_o}{z_F})^{1/2} + \frac{v_o}{w_F} ln(\frac{1 - \frac{v_o}{w_F}}{(\frac{z_o}{z_F})^{1/2} - \frac{v_o}{w_F}}) \tag{5.17}$$

Where z_o is the particle height, again assumed to be the crown of the tree. z_F is the flame height. The height of the flame is found as the vertical height from the ground to the tip of the flame [13]. v_o is the terminal velocity of the particle and w_F is the flame gas velocity. Again, the data for flame gas velocity was not available to the researchers at this time. A workaround to this problem may be achieved by calculating the ratio of v_o/w_F as:

$$v_o/w_F = B(D_p/z_F)^{1/2} \tag{5.18}$$

Where D_p is the particle diameter, and B is a constant set to 40.

The second time section that needs to be calculated t_2 is the time it takes the particle to travel between the flame tip and the buoyant plume and is given by:

$$t_2 = 0.2 + B(\frac{D_p}{z_F})^{1/2}(1 + B(\frac{D_p}{z_F})^{1/2} ln(1 + \frac{1}{1 - (\frac{D_p}{z_F})^{1/2}})) \tag{5.19}$$

and t_3 is defined as the time it takes the particle to complete its ascent through the buoyant plume. It is given by:

$$t_3 = \frac{a_x}{0.8\frac{v_o}{w_F}}(ln(\frac{1-0.8\frac{v_o}{w_F}}{1-0.8r\frac{v_o}{w_F}}) - 0.8\frac{v_o}{w_F}(r-1) - \frac{1}{2}(0.8\frac{v_o}{w_F})^2(r-1)^2) \tag{5.20}$$

Where:

$$r = (\frac{(b_x + \frac{z_o}{z_F})}{a_x})^{1/2}$$

$$a_x = 5.963$$

$$b_x = 4.563$$

a_x and b_x are constants defined by Albini [1] and relate to the flame structure. Albini defines the height to which a fire brand will travel as the point in time where the buoyant flow structure of the tree t_f equals t_t. The buoyant flow time is defined as:

$$t_f = t_o + 1.2 + \frac{a_x}{3}((\frac{b_x + \frac{z_o}{z_F}}{a_x})^{3/2} - 1) \tag{5.21}$$

In this implementation, these values had to be approximated in several ways because not all of the necessary data was available. The maximum height was assumed to be constant, and while the equations are implemented and ready for the correct data, it is not yet available.

A particle will be lofted into the air, and once it reaches its maximum height, the wind then carries it in a direction. If there is no wind in a simulation, the ember would simply fall back to the ground in the cell from which it was launched. Wind is modeled in this library as a 2D vector in each cell of the simulator. The vector contains a magnitude and direction of the wind. The wind vector only contains horizontal data. As in FARSITE's implementation, the fire brand's rate of descent decreases as it falls because as with a real brand, it loses density as it is in the air. Examples of fire brand sizes and their trajectories may be seen in Figure 5.4. The elevation at any time t is given by:

$$z(t) = z(0) - v_o(0)(\frac{t}{\tau} - \frac{1}{2}(\frac{t}{\tau})^2) \tag{5.22}$$

Where:

$$\tau = \frac{4C_d v_o(0)}{K\pi g}$$
$$K = 0.0064$$
$$g = 9.8ms^{-2}$$
$$v_o = (\frac{\pi g \rho_s D_p}{2C_d \rho_a})^{1/2}$$
$$\rho_s = 0.3gcm^{-3}$$
$$\rho_a = 1.2x10^{-3}gcm^{-3}$$
$$C_D = 1.2$$

Where g is the gravitational constant, ρ_s is the density of charred wood cylinder, ρ_a is the density of air, and D_d is the drag coefficient of cylindrical particles.

As the particle descends along its trajectory, its rate of speed in the direction X is determined by the windspeed vector in the cell in which the brand is currently falling. The rate of descent decreases logarithmically as it approaches the top of the tree canopies.

$$\frac{dX}{dt} = U_H \frac{ln(\frac{z}{z_f})}{ln(\frac{H}{z_f})} \tag{5.23}$$

Where z is the current height of the particle, H is the current height of the forest canopy, and z_f is the friction length of the particle, which is set to $0.4306H$. In this implementation of the paper, there is no modeling of 3-dimensional wind, which means that the wind vector does not change based on height. However, in the interest of presenting a complete model for future work, the windspeed at any height H is given as:

$$U_H = \frac{U_{20+H}}{ln(\frac{20+1.18H}{0.43H})} \tag{5.24}$$

In this paper, the method for ignition is modeled by a simple probability which is input by the user. However, there are more complex methods for modeling the ignition of a firebrand which impacts the forest floor. Factors such as forest floor fuel

type, forest canopy filtering, surface moisture content, and the temperature of the fuel [6, 5, 7]. These factors are not examined and are left to future work. Once a fire ignites, it is treated the same as any other ignition source.

In this paper, spotting introduced new challenges to the development of the library. First, it required new data which was not available to the project. Second, the fire simulations, as seen in the previous sections, propagate the fire by testing to see if a cell has already reached the point of ignition. This point of ignition is determined by the value of the time of arrival for the cell being above zero (which represents the initial fire), and below the value which is set as infinity. In practice this value is the maximum value possibly held by a data type. The problem introduced by spotting is that the time of arrival that is calculated by the spotting kernel is a time of arrival that will occur in the future. As the simulation code was written for the previous sections, the fire would begin to propagate the exact next time tick. To fix this issue, the descent of the falling particle is computed at each time tick with the rest of the simulations and stored in a list of falling embers. The list is updated when one burns out or lands to ignite a new fire, or a new ember needs to be added to the list.

If the spotting brand's ignition occurs after a cell has already been lit, then it is ignored. If the brand arrives at a time when a cell is not lit, it lights a fire in that cell, which propagates as a normal fire from that point would.

5.7 Simulation Progression

The simulation has the ability to adapt to the needs of the user. Runtime analysis may be found in Chaper 6, which gives an illustration of the impact of adding more features to a simulation. The more features there are to calculate, the longer the runtime will be. However, it is up to the user of the library to decide what is appropriate for their application. The resolution of the fire and desired propagation features allow the user to use the library to create a simulation capable of a flow much like what is seen again in Algorithm 5.6.

Algorithm 5.6 Simulation Progression

```
InitializeTerrainData();
CalculateSpreadRates();
while Simulation !Complete do
  RunPropagationSimulation();
end while
GenerateOutputFile();
```

The specifics of the library's use are left up to the needs of the user. Results for simulation tests may be seen in Chapter 6.

Chapter 6

Results

6.1 Overview

This chapter presents the results of the testing done on the three propagation methods, their timings, throughput and speedup against the sequential versions. The CPU used to test the implementation was an Intel(R) Core(TM) i7-3770K CPU @ 3.50GHz. The GPU used was a GTX-970 with 4GB of RAM and 1664 cuda cores. The sequential timings were accomplished without implementing any parallel component on the CPU.

In order to run a simulation using this library, three files are needed. A .dem file of the region of interest is needed to determine terrain, and all files are required to be in UTM format. The second file needed is a fuel model file that contains the necessary fuel model data needed for a particular simulation. The third file needed is the fuel model file which relates fuel models to cells in the terrain file. In order to run crowning, three additional data files are needed: canopy height, crown base height, and crown bulk density.

6.2 Spread Images

This section shows the results of the base spread with acceleration integrated for all three methods. The spread patterns produce similar results, but slightly different patterns. The results of the three spread propagations for a flat plane with no wind influence can be seen in Figures 6.1 through 6.3.

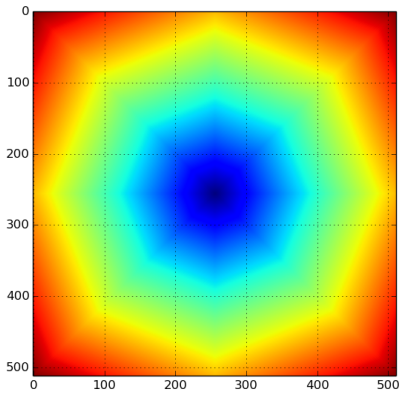

Figure 6.1: The Burn Distances Kernel burn pattern at a grid size of 512x512.

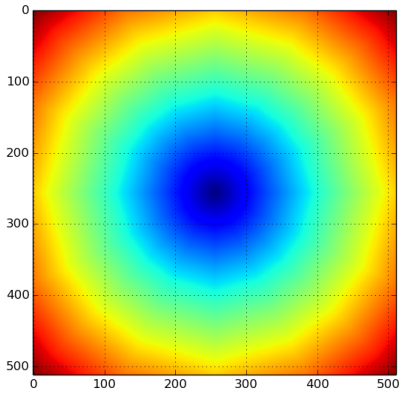

Figure 6.2: The Minimal Time Kernel burn pattern at a grid size of 512x512.

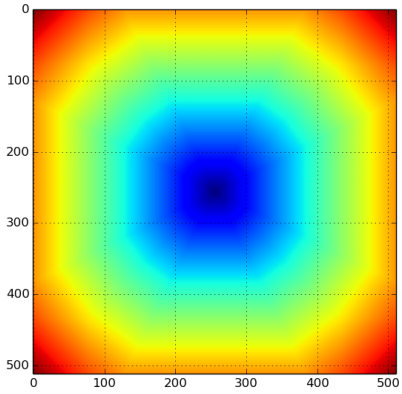

Figure 6.3: The Iterative Minimal Time Kernel burn pattern at a grid size of 512x512.

6.3 Running Time Comparison

6.3.1 Overview

This section compares the running times achieved by the different methods. It first compares the effect of adding the crowning behavior to the simulation and its impact on running time and then presents the runtime results of the experiments.

6.3.1.1 Crowning and Acceleration Time Differences

This section presents results on the running times for the crowning version of the implementation. The base spread method was compared against the crowning method, and the difference in timings was found to be on average less than 1% of the difference between the running times. The only time it appeared to matter was in the sequential implementation. Because the difference between the two runtimes is negligable, no chart is provided to illustrate the differences.

Because the crowning timings did not impact the runtimes significantly, only the runtime results including crowning are included in this paper. The runtime graphs for the timings may be found in Figure 6.4. It is difficult to see the differences in the runtimes of each of the kernels on the linear graph, so a logarithm-scale graph is also included of the same results in Figure 6.5. Each simulation was run on flat ground with no wind factored into the simulation. The spread patterns produced are the same as those seen in Section 6.2.

Take note that the runtimes for both the sequential and parallel versions of the IMT kernel take less time to run. The kernel does not require the use of atomic functions, and therefore runs much faster than the other kernels. The large simulation sizes were not tested for sequential due to the extreme amount of time they would require to compute. The kernels outperform their partner in the sequential implementation at almost all sizes for every kernel type. Notice that the final 2048x2048 grid simulation for the Burn Distances and Minimal Time Kernels are similar in runtimes to the 512x512 runtimes of their sequential counterparts.

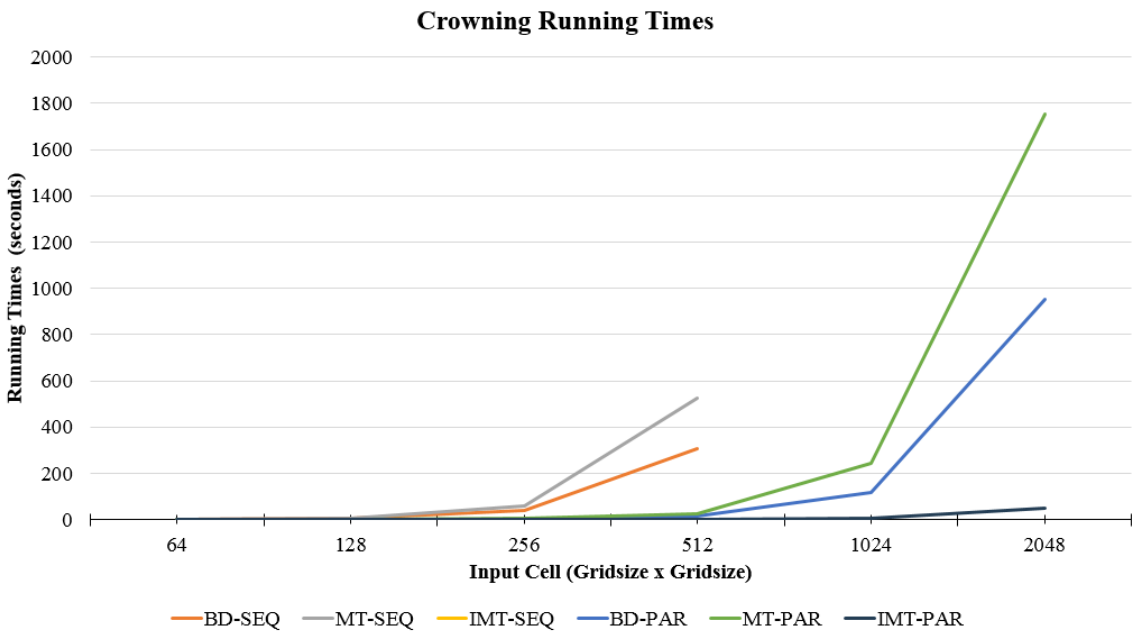

Figure 6.4: The runtimes to completion of simulation

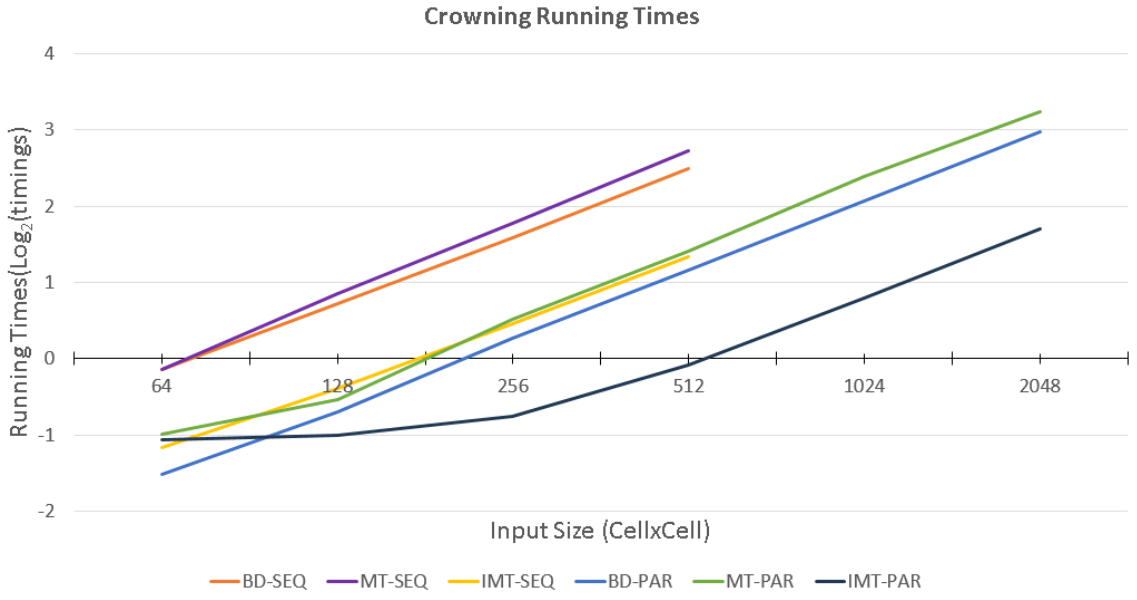

Figure 6.5: The log-scale version of the graph found in Figure 6.4.

6.3.1.2 Throughput and Speedup

Throughput is defined as the amount of data processed per unit time [23]. This paper presents the throughput results for this paper in GigaBytes/Second. The throughput dividing the amount of data processed over the simulation by the amount of time it took to process that simulation. These results are displayed in log_2 scale for ease of comparison in Figure 6.6.

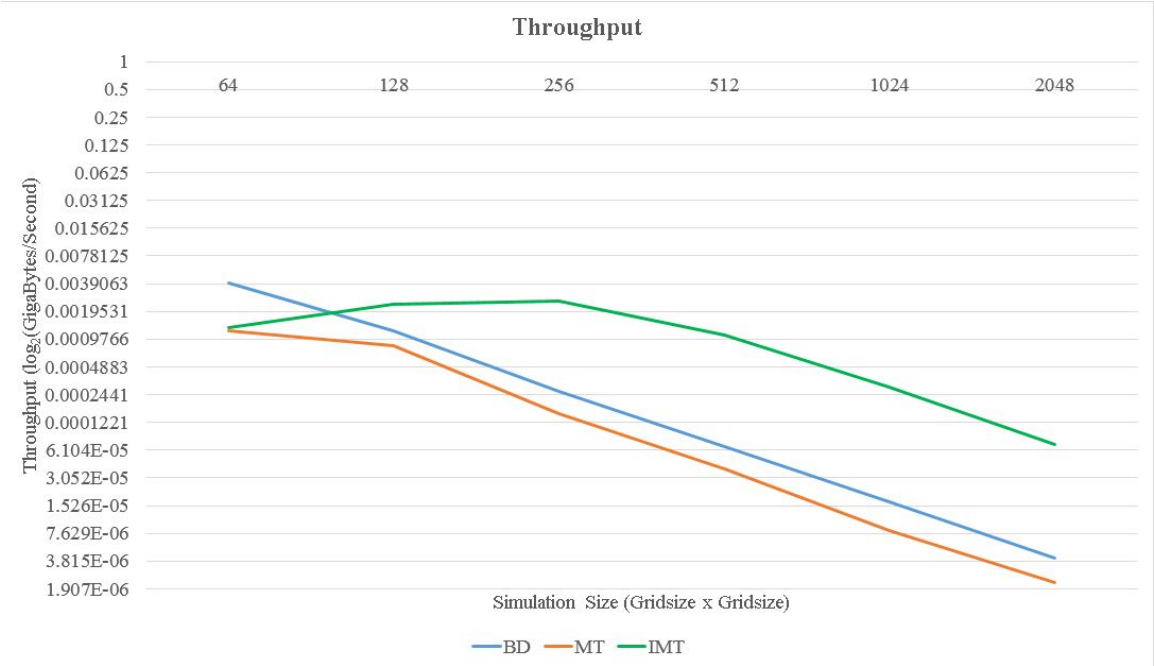

Figure 6.6: The log_2 scale of the throughput results.

The speedup is the amount that the parallel version is faster than the sequential version. This value is found by dividing the sequential running times by the parallel. These results can be seen in Figure 6.7.

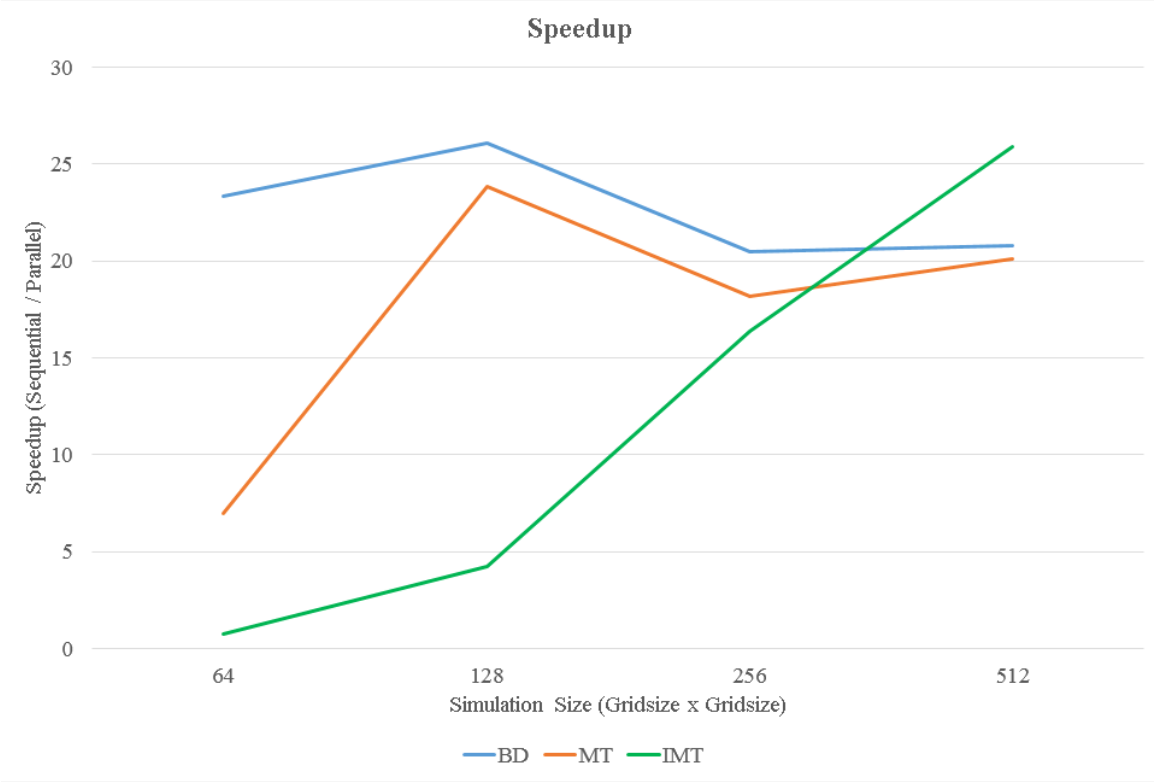

Figure 6.7: The speedup between the parallel and sequential implementations. The average speedup was around 20X faster for all the versions.

6.3.1.3 Memory Transfer Impact

A major part of the time requirements for using CUDA is shipping the data from the host to the device [31]. In order to illustrate this time difference, the average running time with and without memory transfer was timed and may be seen in Figure 6.8. Additionally, the average time it took to transfer memory per simulation size can be seen in Table 6.1.

Table 6.1: The average time to transfer memory from the CPU host to the GPU device in the three largest simulation sizes.

	Average Memory Transfer Time
512	0.09242
1024	0.11389
2048	0.10288

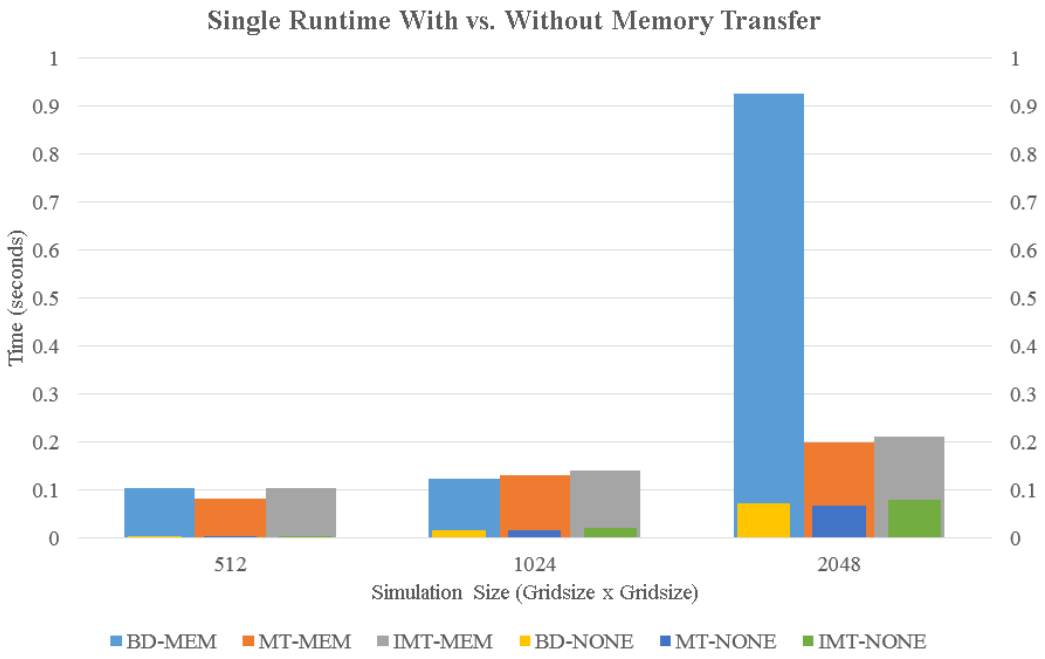

Figure 6.8: The runtimes found for a single iteration of the kernel call with and without memory transfer time.

6.3.1.4 Simulation vs. Real Time

This section presents the relationship between a real-world second and a simulation second. The library runs in faster-than-real-time, which means in a second of real-world time, the library can compute several seconds of simulation time data. The IMT kernel is fastest by far, which makes sense as it does not require any atomic functions to operate. Table 6.2 shows the comparisons for the three methods on two different input sizes on how they compare for simulation time versus real world time.

Table 6.2: A table containing the values for real-time versus simulation timesteps. Multiple simulation seconds are computed per real-world second.

	Size: 512			**Size: 1024**		
Kernel Used	BD	MT	IMT	BD	MT	IMT
Total Time Run (seconds)	1	1	1	3	3	3
Total Iterations	240	240	220	180	200	180
Max Time in Sim	477	534	6804	347	483	7339
Time / Iterations	1.99	2.22	30.93	1.98	2.415	40.77
Real Time / Sim Time	477	534	6804	119	161	2446.33

6.3.1.5 Wind

A brief example of the influence of wind is presented in this section. The two examples are found as follows, and include a wind direction in the positive x direction in Figure 6.9 and both in the positive x and y in Figure 6.10.

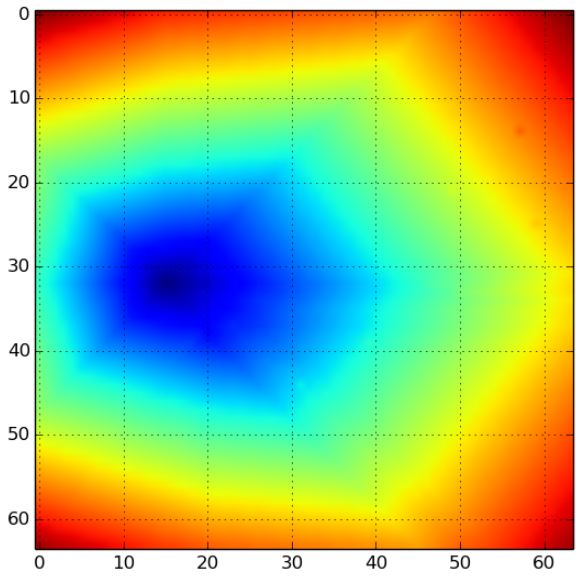

Figure 6.9: Effect of the wind on a kernel when the wind is a value of 40 in the positive x direction.

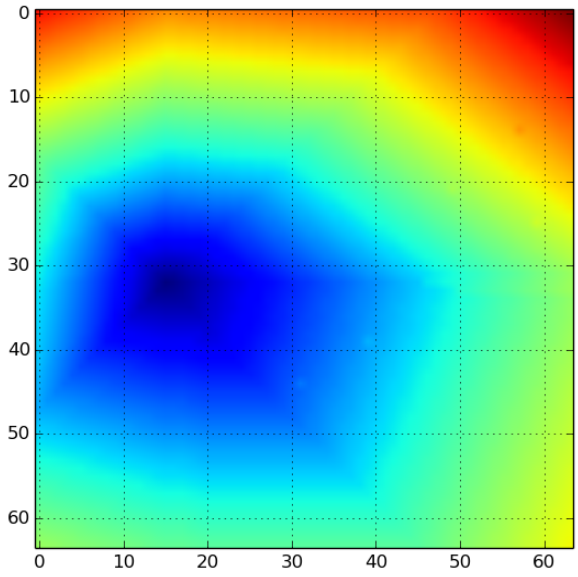

Figure 6.10: Effect of the wind on a kernel when the wind is a value of 40 in the positive x and y direction.

6.3.1.6 Spotting

Unfortunately, the data necessary to run a full-scale spotting test was unavailable. The implementation was performed as though the data was available, however several approximations had to be made. Every ember was assumed to launch to the same height, and then the descent was calculated from that point. The model described in the Implementation chapter was implemented as far as it could go, but for some prototype results, the following figures are presented. Wind is a requirement for spotting to occur, or the ember would be launched and just land back in the already-burning cell. The wind in Figure 6.11 is in the x-direction only, while the wind in Figure 6.12 is higher in the x, but has a value in the y direction as well. The windspeeds were chosen as to be comparable to those presented in Section 6.3.1.5.

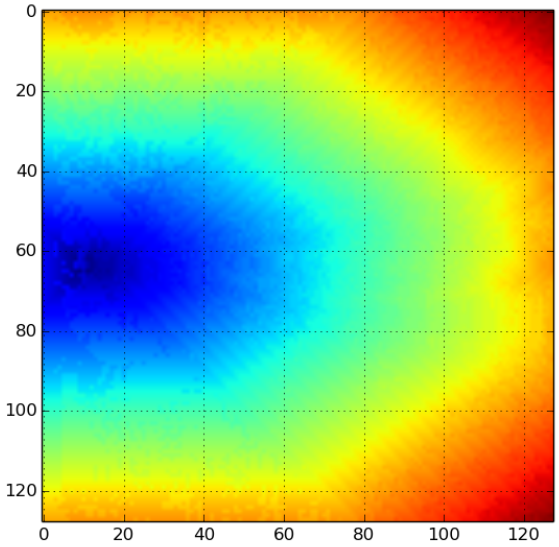

Figure 6.11: The prototype results for the spotting module. The wind in this figure is gusting at a value of 40 in the x direction.

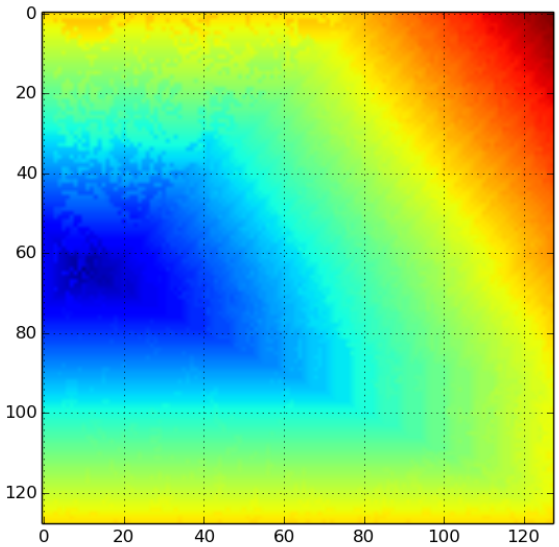

Figure 6.12: The prototype results for the spotting module. The wind in this figure is gusting at a value of 40 in the x direction and 20 in the y direction.

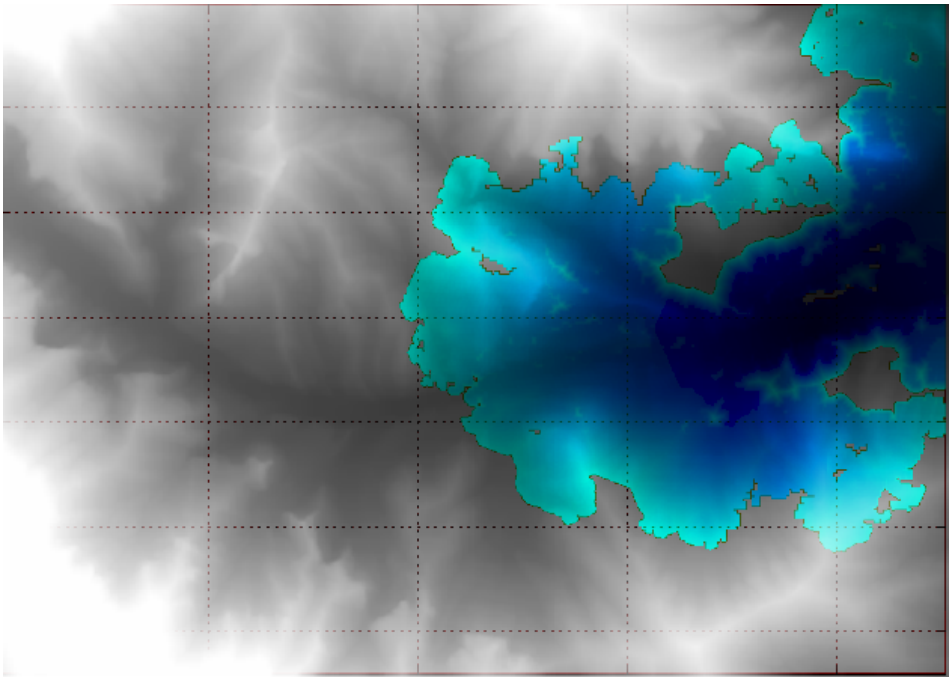

Figure 6.13: The Kyle Canyon fire started at the entrance to the canyon after 5000 iterations in simulation. This simulation is timestamped at approximately 0.68 days or 16.4 hours.

6.4 Kyle Canyon

For the final set of tests, the Burn Distances kernel was run on the data for Kyle Canyon in southern Nevada. Only the BD kernel is shown here because all the outputs are very similar. This is an example of a real-world application of the simulator working. Kyle canyon's Three different points in the simulation are shown: one stopped after 5000 iterations (20.5 seconds to run), the second stopped after 7000 iterations (28.8 seconds), and the third stops after 10,000 iterations (41.1 seconds). The scale at which the simulation was run meant each time step was equivalent to 100 seconds, as was found in vFire [16]. The end time of the simulation was approximately 2.1 days after the beginning of the simulation. The three images are to give a comparison of what the progression of fire looks like.

This thesis presented several avenues for results, including a base spread, running times, throughput comparisons, speedup comparisons, and real-world simulation data.

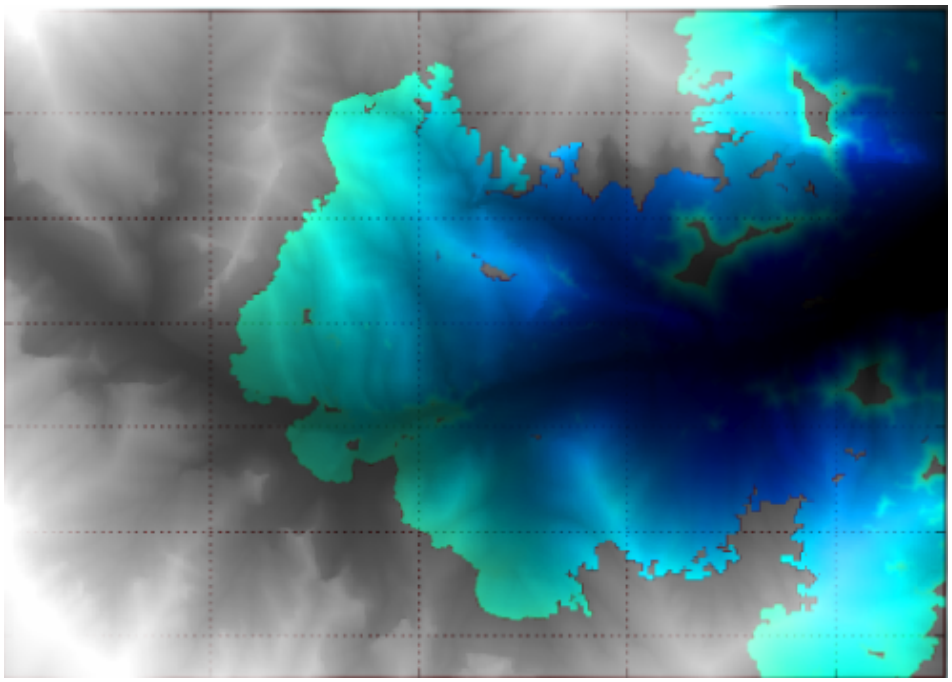

Figure 6.14: The Kyle Canyon fire started at the entrance to the canyon after 7000 iterations in simulation. This simulation is timestamped at approximately 1.2 days.

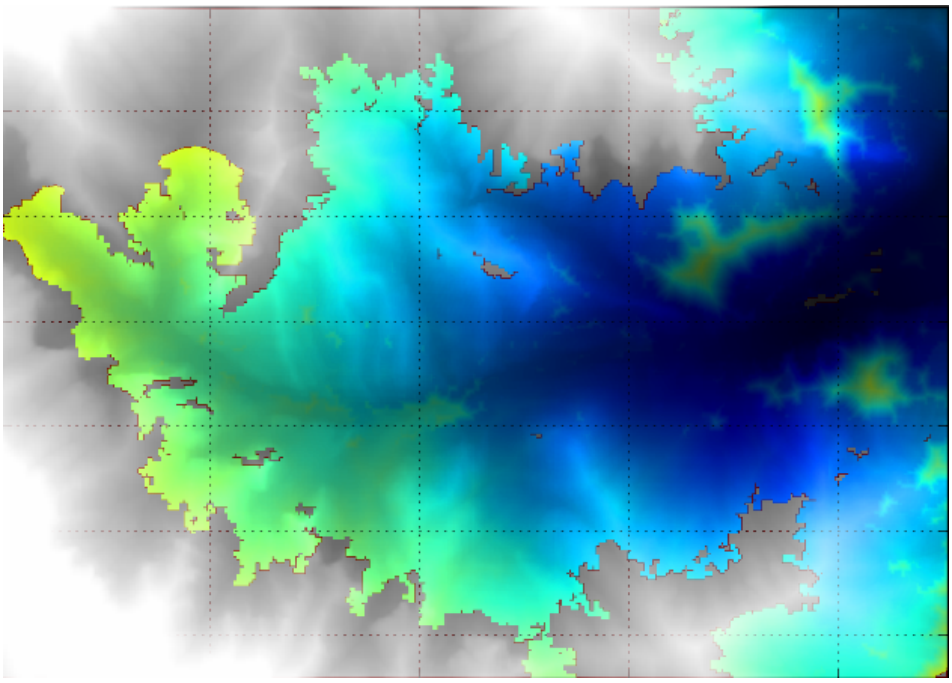

Figure 6.15: The Kyle Canyon fire started at the entrance to the canyon after 10,000 iterations in simulation. This simulation is timestamped at approximately 2.1 days.

The simulation runs in reasonable time on real-world data, and all three kernels achieved approximately 20X speedup. While the work accomplished in this paper is a good start, there are still areas where improvements and additions can be made.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This work implemented vFireLib, a fully-functional forest fire simulation library. This library has the ability to simulate three propagation methods that include fire acceleration, crowning, and spotting. The simulation library offers a simple interface to initialize the simulation, provides a method for easily interfacing with the graphics processing unit, runs a simulation, and outputs a time of arrival map.

7.2 Future Work

While this work provides a fully-functioning forest fire library, there are several areas where it could be expanded in the future.

7.2.1 Improved Spotting

Several approximations were made when implementing the spotting method for this work, as presented in Chapter 5. The limitations were mostly based in data limitations rather than simulation limitations. However, a more accurate method for describing the spotting needs additional data. A more comprehensive model of the atmosphere above a forest fire would increase the accuracy of the spotting method as it would allow for multiple layers of wind vectors to show the impact of wind on the fire itself. The implementation of the spotting calculates a "future burn" when a fire brand ignites a fire ahead of the fire. This map of future burns can be used to queue additional

desired fires by the user, which could allow fire scientists more detailed control over the simulation. It is the recommendation of this work to not base the next implementation on FARSITE, as their model is inconsistent and extremely poorly documented.

Additional work needs to be done on the method for calculating whether a fire brand ignites a new fire at it's point of contact with the forest floor. The current system only uses a user-defined probability, but there are more factors which influence it, such as the type of fuel found on the forest floor, the filtering of the forest canopy, surface moisture content, and the temperature of the fuel [6, 5, 7].

7.2.2 Atmospheric Incorporation

Forest researchers are interested in the carbon footprint caused by a forest fire. This environmental cost is just as important as the physical cost of loss of property and life that can be caused by a wildfire. As a fire burns, it releases a substantial amount of carbon dioxide into the atmosphere. Knowing the impact of a fire could encourage prevention methods or understanding of a previously existing fire on the environment.

With the increase in need for understanding atmospheric impact of a forest fire, a better model of the interaction between the fire and wind data is needed. A wind mesh could increase the accuracy of the spotting method, as mentioned previously, but could also be used to determine the direction the smoke will be blown as well.

7.2.3 Multi-GPU Implementation

CUDA provides the ability to program for multiple GPUs at the same time [23]. The ability to use multiple GPUs to process the simulation could further improve the runtime. Problems that will arise from implementing the library on multiple GPUs include stitching the simulation together at the edges and triggering memory transfer between GPUs at the point where simulations bleed across the cells being processed on individual GPUs.

7.2.4 Visualization

In order to better understand the impacts of forest fires in their environment, a visualization is needed. Visualizing the simulation is an important next step for future work of this project. It would allow an interactive environment in which the simulation could exist and be modified dynamically.

The fire library is built to be used as a tool to run calculations for any program which wished to utilize it. The library should allow for dynamic changes to the terrain values which could simulate real-world firefighting tools such as bulldozing tree lines, water and fire retardant drops, as well as tests to evaluate the effectiveness of fire containment methods by the forest service.

Bibliography

[1] Frank A Albini and Robert G Baughman. Estimating windspeeds for predicting wildland fire behavior. *USDA Forest Service Research Paper INT (USA)*, 1979.

[2] Patricia L Andrews. Behave: fire behavior prediction and fuel modeling system-burn subsystem, part 1. 1986.

[3] Jim Arlow and Ila Neustadt. *UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design*. Pearson Education, 2005.

[4] Collin D Bevins. Firelib user manual and technical reference. *Systems for Environmental Management*, 1996.

[5] Winfred H Blackmarr. Moisture content influences ignitability of slash pine litter. 1972.

[6] Larry S Bradshaw, John E Deeming, Robert E Burgan, and D Jack. The 1978 national fire-danger rating system: technical documentation. 1984.

[7] Stephen C Bunting and Henry A Wright. Ignition capabilities of non-flaming firebrands. *Journal of Forestry*, 72(10):646–649, 1974.

[8] Cubix Corporation. Cubix corporation, xpander rackmount 8 gen3 16-channel (128gbps) pcie slot expansion system and hostengine. http://www.cubix.com/. [Online; accessed 23-Septempber-2015].

[9] creately. Creately Online. http://creately.com/diagram-products. [Online; accessed 26-February-2016].

[10] Mark Arnold Finney. Farsite: Fire area simulator: model development and evaluation. *Intermountain Forest and Range Experiment Station, General Technical Report*, 2004.

[11] GDAL Development Team. *GDAL - Geospatial Data Abstraction Library, Version x.x.x*. Open Source Geospatial Foundation, 2016.

[12] Ross W Gorte and Kelsi Bracmort. Forest fire/wildfire protection. Congressional Research Service, Library of Congress, 2006.

[13] National Wildfire Coordinating Group. *Fire Behavior Field Reference Guide*, volume PMS. 437 edition.

[14] Kelvin G Hirsch et al. *Canadian forest fire behavior prediction (FBP) system: user's guide*, volume 7. 1996.

[15] Roger V Hoang, Joseph D Mahsman, David T Brown, Michael A Penick, Frederick C Harris Jr, and Timothy J Brown. Vfire: virtual fire in realistic environments. In *Virtual Reality Conference, 2008. VR'08. IEEE*, pages 261–262. IEEE, 2008.

[16] Roger V. Hoang, Matthew R. Sgambati, Timothy J. Brown, Daniel S. Coming, and Frederick C. Harris Jr. Vfire: Immersive wildfire simulation and visualization. *Computers & Graphics*, 34(6):655 – 664, 2010.

[17] Roger Viet Hoang. *Wildfire Simulation on the GPU.* ProQuest, 2008.

[18] Eunmo Koo, Rodman R Linn, Patrick J Pagni, and Carleton B Edminster. Modelling firebrand transport in wildfires using higrad/firetec. *International journal of wildland fire*, 21(4):396–417, 2012.

[19] Eunmo Koo, Patrick J Pagni, David R Weise, and John P Woycheese. Firebrands and spotting ignition in large-scale fires. *International Journal of Wildland Fire*, 19(7):818–843, 2010.

[20] RR Linn and FH Harlow. Firetec: a transport description of wildfire behavior. Technical report, Los Alamos National Lab., NM (United States), 1997.

[21] Samuel L Manzello, Alexander Maranghides, and William E Mell. Firebrand generation from burning vegetation. *International Journal of Wildland Fire*, 16(4):458–462, 2007.

[22] RS McAlpine and RH Wakimoto. The acceleration of fire from point source to equilibrium spread. *Forest Science*, 37(5):1314–1337, 1991.

[23] CUDA Nvidia. Programming guide, 2008.

[24] E Pastor, L Zarate, E Planas, and J Arnaldos. Mathematical models and calculation systems for the study of wildland fire behaviour. *Progress in Energy and Combustion Science*, 29(2):139–153, 2003.

[25] Michael A Penick, Roger V Hoang, Frederick C Harris Jr, Sergiu M Dascalu, Timothy J Brown, William R Sherman, and Philip A McDonald. Managing data and computational complexity for immersive wildfire visualization. *Proceedings of high performance computing systems (HPCS'07), Prague, Czech*, 2007.

[26] GLW Perry. Current approaches to modelling the spread of wildland fire: a review. *Progress in Physical Geography*, 22(2):222–245, 1998.

[27] Seth H Peterson, Marco E Morais, Jean M Carlson, Philip E Dennison, Dar A Roberts, Max A Moritz, David R Weise, et al. Using hfire for spatial modeling of fire in shrublands. 2009.

[28] Matt P Plucinski and Wendy R Anderson. Laboratory determination of factors influencing successful point ignition in the litter layer of shrubland vegetation. *International Journal of Wildland Fire*, 17(5):628–637, 2008.

[29] Richard C Rothermel. How to predict the spread and intensity of forest and range fires. *The Bark Beetles, Fuels, and Fire Bibliography*, page 70, 1983.

[30] Richard C Rothermel and Intermountain Forest. A mathematical model for predicting fire spread in wildland fuels. *AUSFS*, 1972.

[31] Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming, Portable Documents*. Addison-Wesley Professional, 2010.

[32] Nicolas Sardoy, Jean-Louis Consalvi, Bernard Porterie, and A Carlos Fernandez-Pello. Modeling transport and combustion of firebrands from burning trees. *Combustion and Flame*, 150(3):151–169, 2007.

[33] Dave Shreiner. *OpenGL Reference Manual: The Official Reference Document to OpenGL, Version 1.2*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 1999.

[34] Ian Sommerville. *Software engineering-9th ed. p. cm.* McGraw-Hill Companies Inc., New York, 2011.

[35] FA Sousa, RJN Dos Reis, and JCF Pereira. Simulation of surface fire fronts using firelib and gpus. *Environmental Modelling & Software*, 38:167–177, 2012.

[36] CE Van Wagner. Conditions for the start and spread of crown fire. *Canadian Journal of Forest Research*, 7(1):23–34, 1977.

[37] CE Van Wagner. Prediction of crown fire behavior in two stands of jack pine. *Canadian Journal of Forest Research*, 23(3):442–449, 1993.