An OpenCL Implementation of the Irregular Terrain with Obstructions Model

Andrew J. Musselman

September 10, 2013

Abstract

This document describes the results of efforts to port the Irregular Terrain with Obstructions Model (ITWOM) for radio-frequency propagation prediction to the OpenCL programming language so that it can be run in parallel. It summarizes the history of the ITWOM and it's predecessor the Longly-Rice Irregular Terrain Model (ITM). The report also describes various approaches that could be taken to parallelize the use of the ITWOM, the modifications necessary to the SPLAT! wrapper program which is an open-source command-line interface for radio frequency propagation modeling and the ITWOM code itself necessary for parallelization. It also describes the results of benchmarking a modified version of the ITWOM and it's wrapper program to measure the performance gain and loss that resulted from parallelization.

Contents

1 Background **2**
- 1.1 General Description . . . 2
- 1.2 The ITWOM and the Longley-Rice Model . . . 3
- 1.3 SPLAT! . . . 3

2 Related Works **3**

3 Program Description **5**
- 3.1 SPLAT!'s Approach to Signal Loss Analysis Over an Area . . . 5
- 3.2 Langley Rice and ITWOM Approach to Path Loss . . . 5
- 3.3 Parallelization Strategies . . . 6
  - 3.3.1 Generate Paths on the Device . . . 6
  - 3.3.2 Path at a Time . . . 6
  - 3.3.3 Block at a Time . . . 6
  - 3.3.4 Multiple Paths/Blocks at a Time With Callbacks . . . 6
  - 3.3.5 Multidimensional Work Items . . . 7
- 3.4 Issues with Parallelization . . . 7

3.4.1 Repeated Points 7
3.4.2 The qlrps() Function 8

4 Implementation and Modifications to SPLAT! and the ITWOM 9
4.1 Changes to the ITWOM 9
4.2 Implementation of Parallelization 9
4.2.1 "Generate all Paths on the Device" 10
4.2.2 "Multidimensional work-items" 10
4.2.3 "Path at a time" 10
4.3 Debugging Techniques 10

5 Benchmark results 11
5.1 SPLAT! Baseline Performance 11
5.1.1 Test Procedure 11
5.1.2 Running Time 12
5.1.3 Running Makeup 13
5.2 Performance Changes with Parallelization 13
5.2.1 Maximum Theoretical Performance Gain 13
5.2.2 Parallel SPLAT! Performance Benchmarks 15
5.3 Verification of Results 17

6 Conclusions 18
6.1 Performance Data 18
6.2 Other Notes 18
6.2.1 OpenCL 18
6.2.2 SPLAT! 19
6.2.3 Potential Intel OpenCL Bugs 20
6.2.4 Copyright Status of ITWOM 21

7 Further Work 22
7.1 ITWOM on GPU and other Hardware 22
7.2 ITWOM and Langley Rice Code Cleanup 23
7.3 Implementing an Improved Wrapper for ITWOM and Langley Rice 23

A Complete Source Code and Data 24

References 24

1 Background

1.1 General Description

Imagine driving through a hilly area while listening to a terrestrial FM broadcaster on the car radio. In many places, a radio listener will have a rich, clear signal. In other places, the reception might be choppy or weak. In some, places

the station might be completely inaudible. This hypothetical example shows a problem that broadcasters and other radio system planners have to deal with: the effects of terrain on signal reception.

Sine the 1960s engineers have used computers to attempt to model the effects of terrain on where a signal can be heard and at what strength.[9] By repeating this process for a number of different paths over an area, its possible to generate a coverage map that shows the theoretical signal strength of a transmitter over a given area. There are many software packages that do just this using many different models, including an open source application called SPLAT! (Signal Propagation, Loss, And Terrain) that is developed by John A. Magliacane.

1.2 The ITWOM and the Longley-Rice Model

The Longley-Rice propagation model was initially developed by Anita G. Langley and Phillip Rice in the 1960s, following a series of studies on the topic [9]. It was initially implemented using the Fortran IV programming language on a Control Data Corporation CDC-3600 [9]. This code was later ported to C by the United States Department of Commerce[12].

The Irregular Terrain With Obstructions Model (ITWOM) is derived from the older Longley-Rice Model, with a number of changes and corrections that its author, Sidney E. Shumate, claims improve its overall accuracy. Other dispute that claim: Alex Farrent, author and maintainer CloudRF, a graphical interface for radio frequency propagation modeling, says that he has "yet to see a proper independent assessment that it is better as claimed"[1]. Shumate designed his code to work as a drop-in replacement for the Longley-Rice Model[10]. Since the two algorithms are so similar in terms of their overall structure and control flow, existing research into how one is parallelized is directly applicable to the other.

1.3 SPLAT!

SPLAT! is a command-line wrapper program for the ITWOM, Longley-Rice Model and other RF-Propagation algorithms. It takes in appropriately formatted elevation data and various parameters about each of the transmitters or receivers that the user wishes to model. It has the option of outputting a bitmap or a KML file for Google Earth for graphically representation coverage information. It also generates text files that contain information about each radio site and nearby obstructions[7]. An example of the type of area coverage plot that SPLAT! generates is shown in Figure 1.

2 Related Works

There are a number of existing serial implementations of the Longley-Rice Model, including a reference implementation maintained by the US Department of Commerce[12]. This reference implementation is the basis of the ITWOM[10].

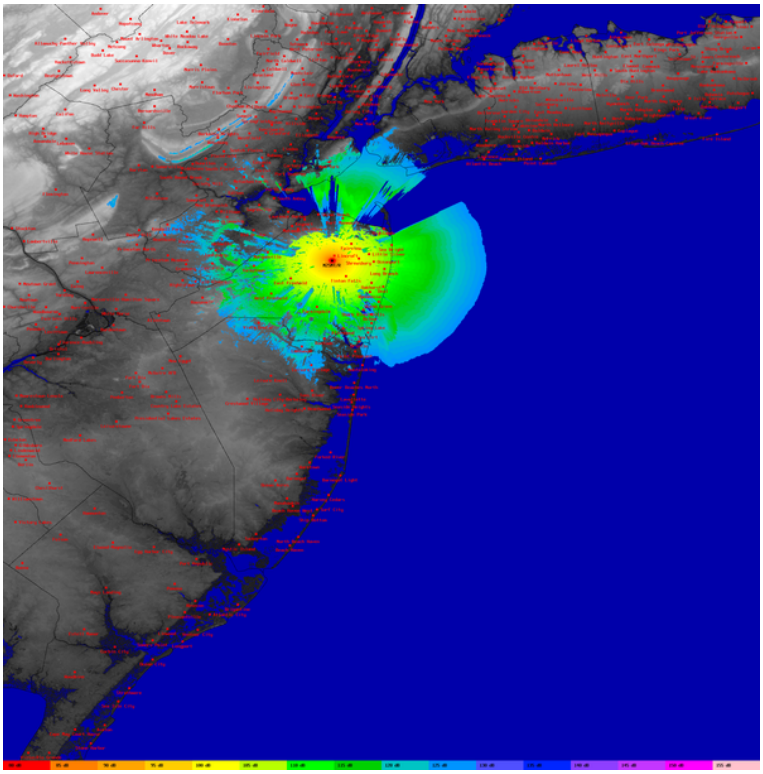


Figure 1: An example of a RF prediction plot generated by SPLAT! Image by John A. Magliacane as retrieved from the SPLAT! website. [7]

Additionally, a pair of University of Arizona researchers have recently made implementations of the Longley-Rice Model for both IBM's Cell processor and for Nvidia's Tesla C870 GPU[11]. These researchers were able to reduce execution times to less than a second for some data sets. They also found that the Tesla GPU was around 2-3 times faster than the IBM cell processor.

Finally, Alex Farrent, maintainer of graphical program for radio frequency terrain modeling called CloudRF has attempted to modify SPLAT! to calculate Longley-Rice path loss in parallel using pthreads. In the process he reported encountering issues with SPLAT!'s extensive global variables. He later successfully implemented process-level parallelization by having two instances of SPLAT! each generate one half of the image and using an external image manipulation library to combine them into one image[1].

3 Program Description

3.1 SPLAT!'s Approach to Signal Loss Analysis Over an Area

After loading the necessary terrain files, SPLAT! loops over all points on the border of a square around the transmitter site. For each point, it generates a path from the transmitter site to the point on the border of this square. This path consists of an array that represents the terrain heights between the transmitter site and the point on the border of the square, along with an overall length of the path and the number of points stored in the path. Then, for each point on all of these paths, it calls ITWOM's point to point path loss function for every possible point between the transmitter and the point on the edge of the analysis area. This point to point path loss function returns theoretical amount of signal loss along that path, which SPLAT! stores in the area's data structure. This is described using psuedocode in Algorithm 1.

Algorithm 1 General description of SPLAT!'s area analysis

```
let terrain data contain the
    elevation data of the area to be analyzed
let path loss data be a structure
    that stores the loss at each point in the area
for each permiter point around the perimiter of
the analysis area
    let path elevation[] = the elevation of all points
        between the transmitter and the perimiter point
    for each point along path elevation
        if path loss data is not already analized at the point
            path loss data at the point =
                point_to_point(elevation[], point, paramters)
            mark path loss data at the point as analized
```

3.2 Langley Rice and ITWOM Approach to Path Loss

Both Langley Rice and the ITWOM have the same overall program structure. In fact, the ITWOM source file that shipped with SPLAT! contained both versions of the Irregular Terrain Model. Many of the functions were shared between them, and the ones that weren't often shared the same parameters and purpose.

Both algorithms first determine if the receiver site is in line of site of transmitter. Then, they call a separate set of calculations for each [9]. As a result, the calculations within the point_to_point() function cannot be easily parallelized.

3.3 Parallelization Strategies

3.3.1 Generate Paths on the Device

One approach to parallelizing this problem is to take the task of generating each terrain profile and put it on the device side. Then, run through a loop on the device side processing each point along each path, putting the results back into the data structure that represents both signal loss and elevation at each point. The advantage to this approach is that the system has to transfer vastly less data to and from the device and it requires less device and host memory.

However, this comes at a cost of greater complexity of the code that is run on the device. Implementation is much more difficult than some of the other approaches because there is that much more code that has to be adapted to run in OpenCL and synchronization of work items that are accessing the same data structure becomes more of an issue. There is also additional complexity in the act of copying the results to and from the device. Since the host memory exists independently of the device memory, it's necessary to write additional code that that makes a deep copy of the terrain and signal loss data structure(s).

The author initially attempted to implement this approach before encountering difficulties, and switching to the multidimensional work-items approach described in Section 3.3.5.

3.3.2 Path at a Time

Another approach is to simply push one path at a time to the device and run the path loss calculations in parallel for each point along that path. This approach has the advantage of having the lowest memory requirements of all of these proposed methods. This comes at a cost of having to transfer much more data to and from the device over the course of plotting the path loss over an area. For points that are close to the transmitter, the distance between two paths is minimal, which means the same altitude data is sent to device code multiple times with this approach. There is also the additional cost in having to enqueue each item separately.

3.3.3 Block at a Time

It's possible to reduce the overhead of the path-at-a-time approach by creating a buffer that contains a certain number of paths and another buffer that is used to store the path loss results from each point. In exchange, both the host system and the device must have the memory to support this buffer.

3.3.4 Multiple Paths/Blocks at a Time With Callbacks

OpenCL also provides a callback functionality that allows host system code to be called when a work item completes. With the right host system code, this could be used load more data onto the device while other elements are calculating. One could even go so far as to have calculations running while more data is

being loaded from disk, so that the system can do path loss calculations while also doing IO. However, the author believes that this would effectively require a complete re-write of SPLAT!, which doesn't currently support any sort of dynamic loading of terrain data at all.

3.3.5 Multidimensional Work Items

OpenCL also provides functionality to deal with multidimensional work items. This gives each device-side work item an address in multiple dimensions, allowing multiple work items to process each path simultaneously. This eliminates the need to loop over elements in a path on the device, which makes the task more parallel. However, this approach still requires sending each path to the device individually, which means that more data must be sent to the device than if the paths were generated on the device itself.

3.4 Issues with Parallelization

For the most part, each path is independent of the others. The results of one point to point calculation are not needed to calculate others. However, there are some issues with the ITWOM code that might effect parallelization.

3.4.1 Repeated Points

The only data dependency is the marking of points as having already been analyzed. Due to the geometry of each path, points closest will be hit multiple times, as shown in Figure 2, so this check is still necessary to prevent duplication of effort. If the "generate paths on device" approach described in Section 3.3.1 is used, it might be possible to avoid this duplication of effort by synchronizing reads and writes to the global data structure. If the "path-at-a-time" approach described in Section 3.3.2, it should be possible to determine which points need to be calculated before sending the data to OpenCL. The host code can then bundle this up as an extra parameter for the OpenCL Kernel, which can then use that parameter to know that the repeated point is repeated and ignore it. It might be simpler to simply throw away the calculations, but this comes at a cost of potentially hurting performance, possibly to the point that performance is worse than without parallelization.

It may also be possible to apply some sort of memoization to the other approaches, but it becomes somewhat more complicated. The multi-dimensional work-item approach described in Section 3.3.5 requires synchronization between multiple device-side work items. In either of the "block-at-a-time" approaches described in Section 3.3.3 and Section 3.3.4 code must check both against SPLAT!'s host-side count of which points have been calculated and with each other. Any approach that involves multiple host-side threads, such as the approach involving callbacks described in Section 3.3.4 requires synchronization between these host-side client threads, in addition to any device-side synchronization.

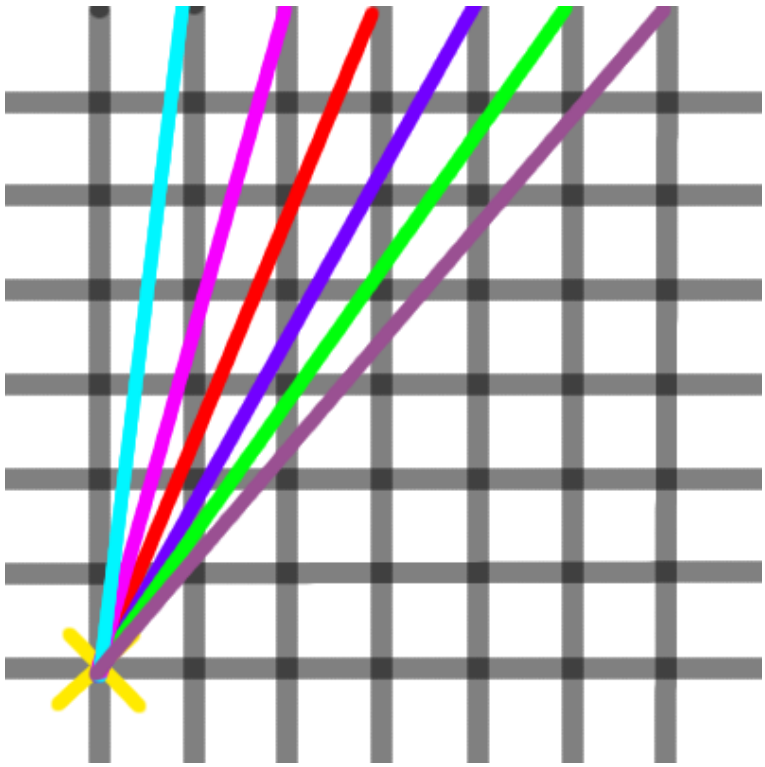


Figure 2: Graphic representing how paths converge near the transmitter. Each intersection in the grid contains variables for both an elevation value for input and a signal loss value as output. The transmitter site is represented with a yellow 'X'. Note how some points, like the one that is two rows up and one column to the right of the transmitter, are close to multiple paths.

3.4.2 The qlrps() Function

Both the Longley-Rice and ITWOM ITM algorithms use a function called qlrps() to initialize several parts of an internal data structure. This function does not take the terrain profile or any other path specific parameters as inputs. Since all the parameters it takes as input are the same for all paths that SPLAT! is calculating, this results in the same code being repeated multiple times with the same inputs, which might be considered a waste of resources. The University of Arizona researchers who were investigating a parallel implementation of Longley-Rice modified the program so that this initialization step was only needed once [11]. To simplify implementation and to minimize the effects of modifications which are not necessarily related to parallelization the author choose not to modify this function beyond what was necessary run it in OpenCL. This means that the qlrps() function is called multiple times on the OpenCL device.

4 Implementation and Modifications to SPLAT! and the ITWOM

4.1 Changes to the ITWOM

Regardless of which overall approach is taken to parallelization, the ITWOM code must be modified to allow it to even compile in OpenCL. The ITWOM code contains many instances of static and it often uses a special call to initialize a function's static local variables before calling the same function again to execute the actual calculations needed to produce a projected signal loss. OpenCL does not support the static keyword. Therefore the author had to modify the ITWOM to store these static variables in a struct that is kept as a local variable in the calling function. Whenever one of these functions is called that once contained a static variable is called, it is passed to a pointer to an instance of the struct that is declared in the calling function.

Some parts of the ITWOM use the C++ Standard Template Library and complex number library. Neither of these exist in the Intel OpenCL implementation that the author was using for the bulk of this project. As a result, the author had to implement his own complex number type and a number of complex number mathematical operations.

The hzns2() function in the ITWOM has some statements that take the result of some mathematical calculations and attempts to use it as an array index. In rare instances, this can result in an array index that is out of bounds, in turn resulting in a segmentation fault when the program attempts to access this out-of-bounds memory. The author added a simple bounds check to prevent the program from attempting to access a piece of undefined memory. The author also suspects that the case that caused a segfault with this line would have normally been suppressed by SPLAT!'s check for repeated points. As a result, he also suspects that implementing one of the memorization schemes described in Section 3.4.1 might also have avoided this segfault, at least with the specific test conditions he used for his benchmarks.

4.2 Implementation of Parallelization

Any use of Open CL requires additional code to compile the device side code, set up any device-side buffers and potentially populate them with data, set the kernel arguments and read data from from the device. These steps would add some overhead to the process, which is an unavoidable cost of using OpenCL for parallelization. This might put OpenCL at a disadvantage to other parallelization techniques on some systems.

The other changes that are necessary to SPLAT! and the ITWOM depend on which overall approach to parallelization a programmer takes. Through the course of the project, the author attempted to implement the "generate paths on device" strategy as described in Section 3.3.1, the "multidimensional work-item" strategy as described in Section 3.3.5, and the "path at a time" strategy as described in Section 3.3.2.

4.2.1 "Generate all Paths on the Device"

The author initially attempted to implement the ITWOM in parallel by generating all paths on the device, as decried in Section 3.3.1. However this implementation did not work. The author attributes the failure of this method to a combination of the potential compiler bug mentioned in Section 6.2.3 and the difficulty of passing the complex data structure that contains elevation data from the host to the device. Because the host and device have completely separate memory spaces, pointers that are valid in the host process will not be valid in any of the device's memory spaces unless they have been explicitly set as a kernel argument.

Due to the structure of SPLAT! and some of the limitations of OpenCL, a number of features would to be removed from SPLAT! in order to make it work using OpenCL using this approach. Because OpenCL lacks a direct file input and output functionality, the author decided to remove a feature that allowed storing of signal loss results in a log file in addition to being displayed in the final output. Had SPLAT! handled the output of the contents of this log file differently, this measure would not have been necessary. Additionally, the author believes it would be necessary to a feature that computes loss based vertical radiation pattern of an antenna to avoid triggering a potential Intel OpenCL compiler bug.

4.2.2 "Multidimensional work-items"

The author then attempted the multidimensional work-item approach described in Section 3.3.5. In this process, he encountered an unknown crash in the device side code. As a result he, abandoned this approach in favor of the easier to implement "path at a time" approach.

Like an implementation of "generate all paths on the device" approach mentioned in Section 4.2.1, the author believes this approach would have required removing the signal loss result log file and the ability to account for antenna elevation patterns in signal loss calculations for the same reasons.

4.2.3 "Path at a time"

The author then successful implemented the "path at a time approach" as described in Section 3.3.2. Since this approach retains the per-path client-side code from SPLAT!, it is possible to retain the signal loss result log file and the ability to account for antenna elevation patterns in signal loss calculations. This is the approach used for all of the benchmark results described in Section 5.

4.3 Debugging Techniques

Although newer versions of the Intel OpenCL tools support debugging, the version of the Intel OpenCL tools that the author used for the majority of this project does not have any means to use a debugger on the device side code. [5] While there is a printf function available as an OpenCL extension in many

implementations of OpenCL, these implementations often do not function as might be expected. The author found that Intel's printf function would not correctly output variables of the type double. This required the author to take a few creative approaches to debugging.

One technique for troubleshooting a crash was to simply comment out code until the program started working. This was one case where the printf extension was genuinely useful: it would show which code had actually run. To find more subtle errors, the author wrote a simple single-path test program that passed a single path both an unmodified version point_to_point and the OpenCL implementation thereof and compare the results. This code revealed a number of discrepancies under certain circumstances. However, finding the source of these discrepancies within the functions called by point_to_point requires more effort.

Since OpenCL is derived from the C programming language, the author was able to compile the modified version of the ITWOM as host side code, allowing him to run the usual debugging tools on it. To find the source of a numerical discrepancy, he would add a number of printf statements into same points in both the stock version of ITWOM and the modified version that had been translated back to C from OpenCL. Then, he would run both sets of code with the same inputs and compare the output from debug printf statements between both versions of the code.

5 Benchmark results

5.1 SPLAT! Baseline Performance

5.1.1 Test Procedure

To establish a baseline benchmark for SPLAT!, the author ran a set number tests on the existing SPLAT! implementation. For these tests, the author modified SPLAT!'s build script to embed debug information and generate runtime profiles, which may have hurt overall performance slightly.

These tests focused on the terrain surrounding Cal Poly, San Luis Obispo. SPLAT! does not load terrain data for ocean areas. Therefore, running SPLAT! over a coastal area like the terrain surrounding Cal Poly would result in SPLAT! spending less time reading terrain data from disk compared to inland areas. Additionally, all of these tests assume a receiver at 10 feet above ground.

All of these tests were run on Tesla, a Cal Poly Computer Science department server. This computer is equipped with a pair of Intel Xeon E5504 CPUs, each of which have 4 CPU cores[6]. As a result, the machine has a total of 8 CPU cores. The build of SPLAT! used was configured to analyze up to 25 "pages" of terrain, each of which is one degree by one degree wide. Additionally, the author elected to use the highest-resolution elevation data that is available for use with SPLAT!, which has data points around 1 arc second apart.

The author created a simple shell script to run these tests. This script ran as a loop: for each integer between 1 and 100, the script told SPLAT! to simulate

propagation of a given frequency from 50 foot high antenna on the Cal Poly campus to a hypothetical receiver that was located 10 feet off of the ground for all points within that integer miles radius of the transmitter. This would show how total runtime and the number of times specific functions were called were effected by the amount of data to be processed. This shell script is shown in Algorithm 2.

Algorithm 2 Shell script used to run tests. The options given to SPLAT! are: -d for the directory that contains the elevation data, -L for the height of the receiver (in feet), -R for the radius in miles and -t for a file that describes the transmitter's location.

```
#!/bin/bash
for((i=1; i<=100; i++))
do
  echo "Run with $i miles:" > ./data/time_$i.out
  (time
   ../splat-1.4.0/splat-hd
          -d ../eledata/sdf/
          -L 10
          -R $i
          -t W6BHZ.qth)
       1>./data/stdout_$i.out
       2>> ./data/time_$i.out
  echo "Run with $i miles:" > ./data/gprof_$i.out
  gprof ../splat-1.4.0/splat-hd >> ./data/gprof_$i.out
done
```

5.1.2 Running Time

When comparing the radius to the execution time, The author found that the execution time increases polynomially with the radius of the area. This relationship is shown in Figure 3.

However, the amount of terrain data that SPLAT! has to load and process also increases polynomially. Therefore, it makes sense to compare the execution time to the total area over which SPLAT! is calculating signal strengths. This relationship is showed in the graph in Figure 4 on page 14.

Beyond around 10,000 square miles, the execution seems to be growing only slightly faster than linearly with area. The author suspects this gradual increase in the rate that execution time is increasing by time is caused by the overhead of loading additional map tiles and stitching the results of the ITWOM into a completed graph. This would mean that the ITWOM has an experimental O(n), where n is the number of points to be analyzed.

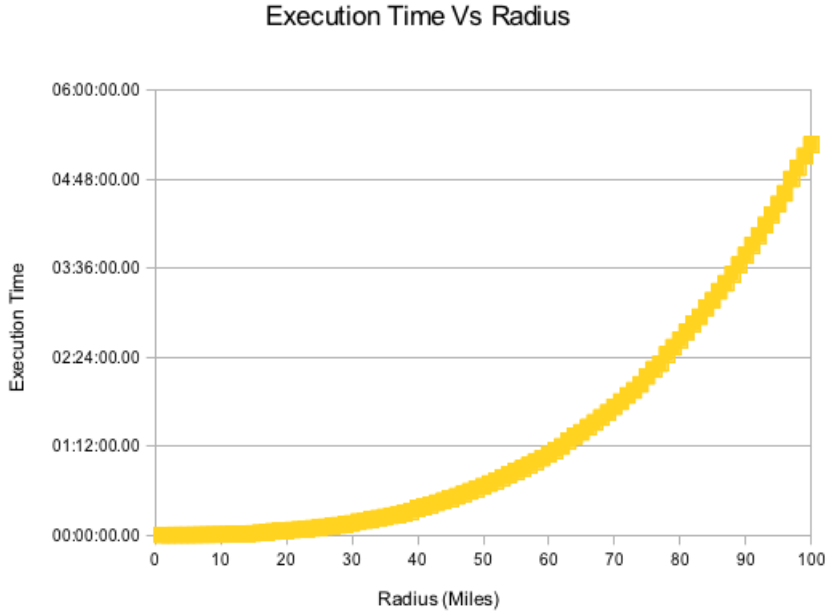


Figure 3: Graph of execution time versus the radius of the area being analyzed.

5.1.3 Running Makeup

For all but the smallest areas, the vast majority of SPLAT!'s computing time is spent running the ITWOM calculations. Generally, as the radius of analysis increases, so does the proportion of time that SPLAT! spends on the ITWOM calculations. However, there are some points where the proportion of time SPLAT spends on the ITWOM calculations briefly decreases. This pattern is shown in Figure 5.

5.2 Performance Changes with Parallelization

5.2.1 Maximum Theoretical Performance Gain

Since the signal loss is at each point is calculated independently of the others, we might be able to estimate the maximum theoretical performance gain by simply dividing the time spent on the ITWOM operations by the number of equivalent compute cores available. This approach is unrealistically optimistic in that it ignores any overhead associated with compiling the OpenCL code, copying data to and from each device, and copying output data into SPLAT!'s global data structures. Nor does it account for any performance loss due to changes that may be necessary due to device or host memory limitations or the increased use of memory on the host. This approach is only valid when

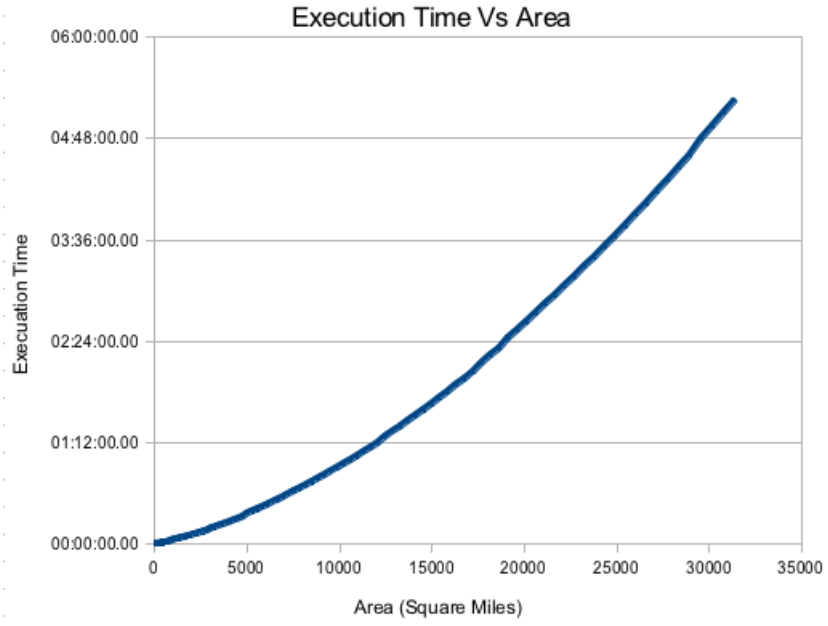


Figure 4: Execution time versus total area that is being analyzed.

comparing single-thread performance to a number of equivalent processor cores. For example, the differing performance characteristics of a CPU and a GPU might make this approach invalid for estimating the maximum performance of SPLAT! if its ITWOM operations were to be run on a GPU.

Based on the data gathered by the benchmark described in Section 5.1.1, if all 8 CPU cores on Tesla could be used for the ITWOM calculations, the time it takes to plot path losses over an an area with a radius of 100 miles could be decreased form over 12,500 seconds to roughly 2,100 seconds. This is a 83% improvement.

Further incremental performance improvements might be possible by adding additional cores. If there are 14,400 cores, that is, one for every path that is generated from the terrain data, than the time it takes complete this task drops to around 590 seconds, for a 95% improvement versus the stock time or a 28% improvement over the 8 core version. However, clustering and networking would be required to to allocate 14,400 CPU Cores to the task. This overhead would likely eat into any further performance gains.

The theoretical running times calculated by this method are shown in Figure 7 and the theoretical performance gains are shown in Figure 6.

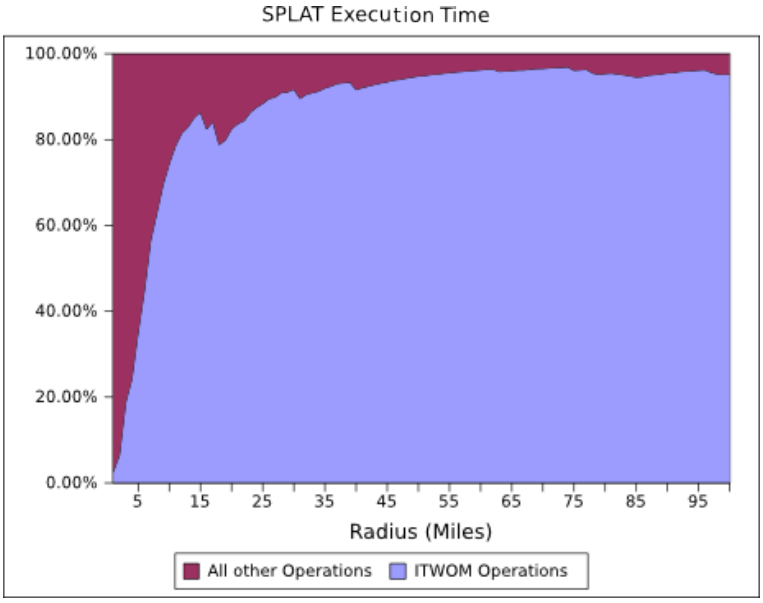


Figure 5: A graph showing the how much of the total execution time of SPLAT! is dedicated to the ITWOM path loss calculations as related to the maximum radius of each given path.

5.2.2 Parallel SPLAT! Performance Benchmarks

To test the change in performance of the parallelized version of SPLAT!, the author ran a series of benchmarks similar to the initial benchmarks described in Section5.1.1 on both the stock version of SPLAT! and a number of modified versions. Unlike the test described in Section 5.1.1, both the stock and modified versions of SPLAT! were compiled to use only 4 "pages" of terrain data. This is due to problems that the author encountered with his modifications with builds that analyze more terrain. The author suspects that this is due to memory limitations of the OpenCL implementations he used. Like the stock benchmarks, the author ran these tests on Tesla, the Cal Poly Computer Science department server. All the OpenCL code was run with the Intel SDK for OpenCL Applications XE 2013, version 3.0 on the server's CPU.

The author benchmarked 3 different versions of the parallelized code in addition to the stock version SPLAT!: One that calculated the entire path without regard to the distance parameter given to SPLAT!, one that only calculates path points out to the given radius, and a third that attempts to avoid re-calculating points that had been already been calculated through memoization. The results of these benchmarks are shown in figure 8.

Overall, all of the parallelized versions that respect the distance parameter showed significantly improved performance at radii longer than approximately

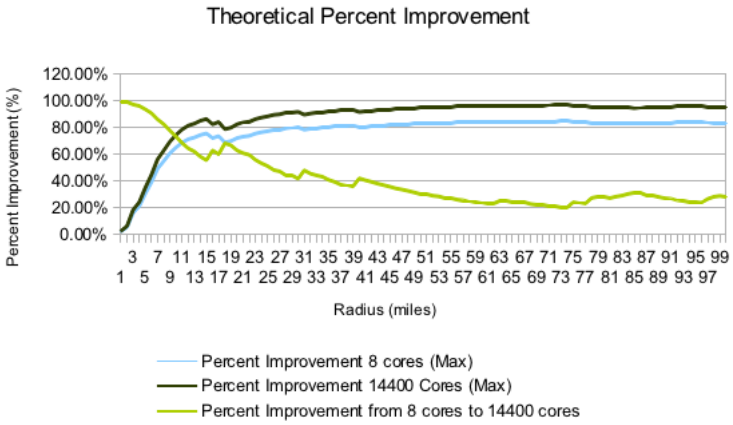


Figure 6: Chart showing the theoretical performance improvement with 8 and 14400 CPU cores assigned to the ITWOM calculations within SPLAT!.

30 miles. At 100 miles radius, the memorized version of the code is nearly 4 times as fast as the stock version. This relationship is shown in figure 9.

The data shows several other interesting trends:

- Unlike the previous benchmark, the run time of stock version starts to increase more slowly at longer radii. This is most likely due to smaller smaller maximum area of analysis. The modified versions show a somewhat similar trend.
- For all radii longer than 53 miles, all of the modified versions had improvements over the stock version of SPLAT! This even includes the version that does not respect the radius parameter and calculates unnecessary additional points.
- Even the modified version that does not respect distance has a change in execution time at a radius of around 16 miles. This is more likely due to SPLAT! loading additional terrain data at that point.
- At very short radii, the all of the parallelization efforts are worse than the stock. This is likely due to the overhead of loading and compiling the OpenCL code.
- All of the parallelized versions have significantly more "User" CPU time reported by the "time" command than overall time. This suggest that time is counting time on each CPU core towards the total "User" time. This is why the author elected to compare overall wall time for this benchmark.

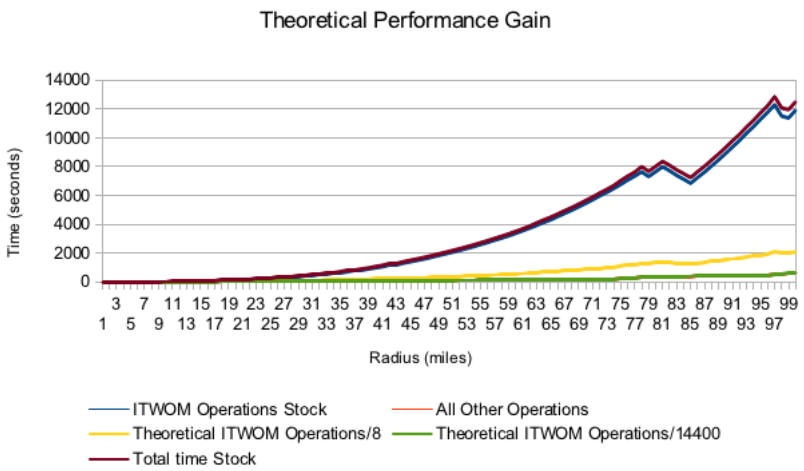


Figure 7: Graph displaying theoretical run times for a parallelized version of SPLAT! along side actual benchmark results

5.3 Verification of Results

As mentioned in Section 4.3, the author wrote a single-path test driver to verify functionality of the ITWOM code. He used this to verify the correct output of the ITWOM with a verity of generated terrain, including terrain that has multiple peaks or valleys, has a single peak or valley or is completely flat. He also tested the ITWOM with a verity of frequencies, transmitter and receiver heights and distance between points. These tests revealed a number of problems with the initial implementation of the complex math functions mentioned in Section 4.1. These tests also revealed an issue with very low-resolution terrain data, that the author chose to ignore because SPLAT! uses higher resolution terrain data. A technical report from Sid Shumate states:

> This computation proved to be unstable and problem-causing beyond a distance of 67 kilometers, and was reburied, as it is not used for high-definition terrain databases, and has been inactive since at least 1967 [2, 250]

Despite these efforts, the author discovered some discrepancies in the overall output of SPLAT! in both the memoized and non-memoized versions. Some small minority of paths in the non-memoized version show vastly less loss than the stock version. The author believes this is due to the bounds-check modification mentioned in Section 4.1. In the memoized version, the author noticed that the corner paths were not being calculated, leaving four small slivers where no loss data was calculated. These defects are shown in figure 10.

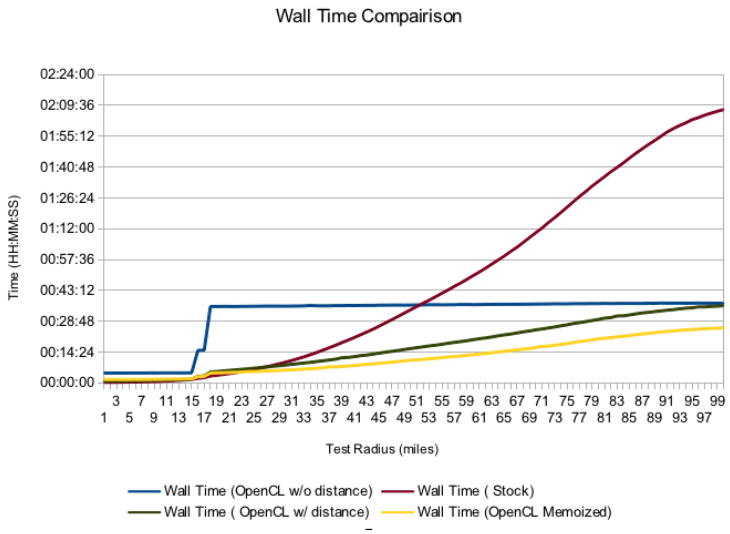


Figure 8: Chart showing the overall runtime for the stock and modified versions of SPLAT!.

6 Conclusions

6.1 Performance Data

The results of this project show that there are significant performance gains to be had by parallelizing SPLAT!. While OpenCL has it's drawbacks and adds some amount of overhead to the process, an OpenCL implementation of the ITWOM running on a CPU is an effective way of parallelizing SPLAT!'s signal loss mapping functionality. With large enough area of analysis and 8 CPU cores, this approach can cut running time by as much as nearly 400%.

The results for longer radii come close to or even beat the earlier numbers that the author calculated as a theoretical upper bound of performance in Section 5.2.1. This relationship is shown in Figure 11. It's worth noting that this may be in part due to the fact the theoretical numbers were derived from a benchmark that was run on a version of SPLAT! which was compiled to use a larger maximum area of analysis. With paths near 100 miles long, the actual benchmark results may be the result of calculating fewer points than stock benchmark calculated, which might explain some of the discrepancy.

6.2 Other Notes

6.2.1 OpenCL

The Khronos Group, who maintains the OpenCL specification, describes OpenCL as the "first open, royalty-free standard for general-purpose parallel program-

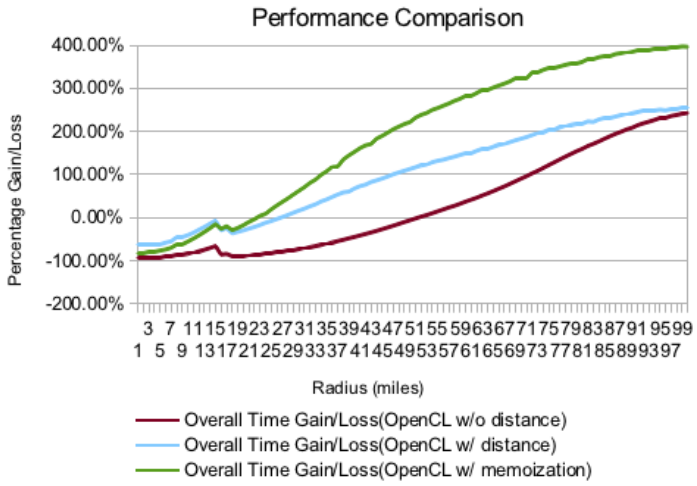
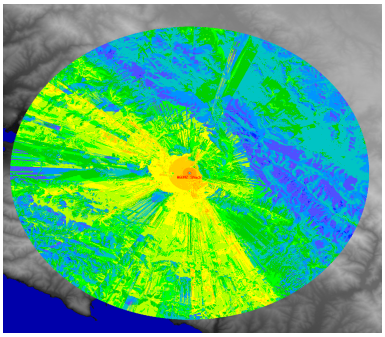


Figure 9: Performance comparison of modified versions of SPLAT!.

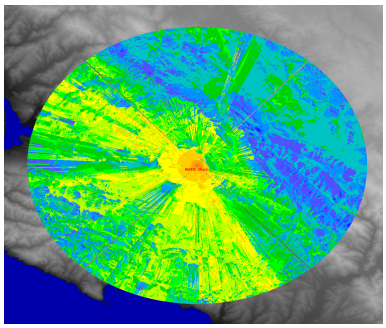
ming of heterogeneous systems. OpenCL provides a uniform programming environment for software developers to write efficient, portable code for high-performance compute servers, desktop computer systems and hand-held devices using a diverse mix of multi-core CPUs, GPUs, Cell-type architectures and other parallel processors such as DSPs[8]". In an ideal world, it should be possible to write code once and run it without modification across multiple types of device. The inherent differences in architecture between different device types mean that "true platform independence is still a goal (versus a reality)"[4, 41]. Beyond that, the author has himself found that some implementations of OpenCL may be incompatible with each other to the extent that code that runs on one may not run on the other.

6.2.2 SPLAT!

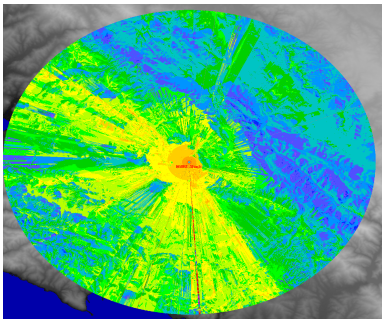
In the process of adapting SPLAT! to use OpenCL, the author found SPLAT! to be difficult to use and almost unmaintainable. With the exception of the ITWOM and Longley-Rice code, all of SPLAT!'s functionality is contained in a single file. SPLAT! uses global variables extensively, making it difficult to re-factor or re-use SPLAT!'s code or to attempt to parallelize it. Additionally, since SPLAT! loads all of the necessary terrain data at once, it has high memory usage.



(a) Stock Signal Loss Plot



(b) Memoized OpenCL Signal Loss Plot



(c) Non-Memoized OpenCL Signal Loss Plot

Figure 10: Comparison of SPLAT! Output. Notice how some of the paths in the plot generated by the non-memoized version of the OpenCL modifications has a few small slivers of paths where SPLAT! has reported a far higher signal intensity than the stock version. Also note that the "corner paths" are not calculated at all in the version of the OpenCL modifications that does implement memoization.

6.2.3 Potential Intel OpenCL Bugs

As mentioned in Section 4, the author believes that he might have found a bug in the OpenCL Compiler. In the device-side code there is a function called PlotLRPath(), which is described in Algorithm 3. The mere presence of certain lines within that function causes any kernel that is defined in the same file to fail, regardless of rather kernel calls the suspect function or not. As stated in Section 4, the author was able to avoid the issue by simply removing the suspect line and all other calculations necessary to account for the antenna's vertical radiation pattern.

The author also found and was able to find and actually isolate a separate and possibly unrelated bug in the Intel OpenCL compiler. In this case, passing a variable that is a struct that contains a two dimensional array to a function

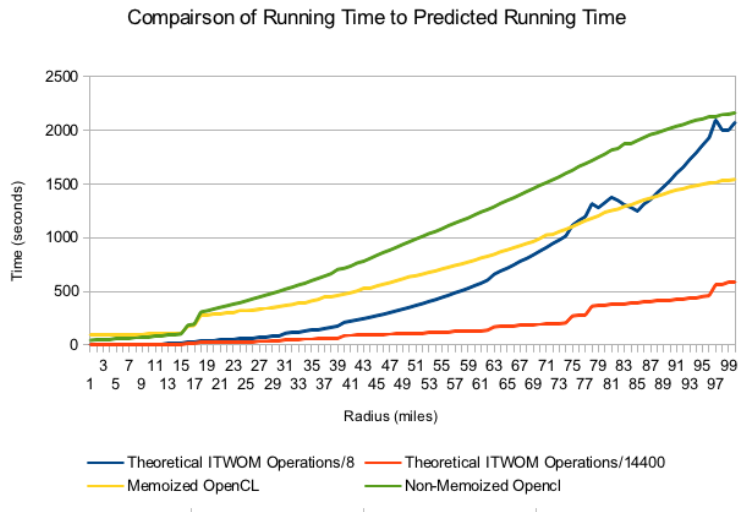


Figure 11: Comparison of Theoretical running times to actual benchmark results

causes the causes the Intel OpenCL compiler to fail silently, resulting in a device-side program that fails to run at all. This failure happens even if the code that contains the two dimensional array isn't called by the kernel that is run. The mere presence of a function call that includes a struct that contains a two dimensional array is sufficient. The author found that the issue did not exist with AMD's Open CL compiler and libraries.

6.2.4 Copyright Status of ITWOM

In the header file of the ITWOM.3.0.cpp file that comes with SPLAT! there is a notice which reads:

> This file is copyright(c) 2011 by Sid Shumate and Givens & Bell, Inc. All rights reserved. Commercial use, and resale, including when compiled with wrap-around software, is prohibited except under Givens & Bell, Inc. license.

This seems to conflict with SPLAT!'s licensing under the terms of the GNU GPL version 2 which specifically allows charging a fee to for transferring a copy in section 1. Section 2 of the GPL v2 specifically requires programs that distributed with any component that is covered by the GPL be covered in whole by the GPL and section 6 forbids anyone from adding restrictions to GPLed code.[3] Since John A. Magliacane distributes SPLAT! with the ITWOM built into it and doesn't expressibly state any exception to the GPL for the ITWOM, we cannot be entirely sure what the license status of the ITWOM is based upon

Algorithm 3 A portion of the PlotLRPath() function that is believed to trigger a failure due to a potential bug in the Intel OpenCL Compiler

```
void PlotLRPath(
        struct site source,
        double altitude,
        struct LR LR,
        const path_t path,
        const double clutter,
        const double max_range,
        const unsigned char got_elevation_pattern,
        const unsigned char dbm,
        int path_point,
        double *loss_result)
{ /* This function plots the RF path loss between source
 and destination points based on the ITWOM propagation model,
 taking into account antenna pattern data, if available. */
...

if (block){
        elevation=((acos(cos_test_angle))/DEG2RAD)-90.0;
}
else {
        elevation=((acos(cos_rcvr_angle))/DEG2RAD)-90.0;
}
...
}
```

the SPLAT! source code itself. While this does not impose any restrictions on the author's work, it has caused confusion amongst other users. For example, Alex Farrent does not include ITWOM functionality in CloudRF due to fear of running afoul of the language prohibiting commercial use. [1]

7 Further Work

7.1 ITWOM on GPU and other Hardware

Through this project, the author did not attempt to optimize his version of the ITWOM for a GPU or any other platform supported by OpenCL. Although there may be significant changes that must be made to allow the code to run properly on a GPU or other platform, there may be significant performance gains in doing so [4, 41].

7.2 ITWOM and Langley Rice Code Cleanup

As mentioned in Section 1.2, the code that implements the Longley-Rice model and the ITWOM have had a long history of different entities modifying it. In fact, Anita G. Langley and Phillip Rice's paper describing the model warns, "the program was written for a Control Data Corporation CDC-3600 computer and may require slight modification for use with other computers" [9]. Through this long history of modifications, many practices that many modern developers would consider to be archaic have remained. For example, the terrain profile array used in the Longly-Rice code stores the number of elements in the array at the first index in the array and the distance between points in the array as the second index in the array. Additionally, most of the comments and documentation that appears in the original Longley-Rice report have been stripped from the newer versions. Correcting these issues would greatly improve the maintainability of the Longley-Rice code as well as any wrapper program that uses it.

7.3 Implementing an Improved Wrapper for ITWOM and Langley Rice

As previously noted, SPLAT! has a number of issues that make it a very difficult piece of software to maintain and work with. It should be straightforward to create a vastly improved set of "wrapper" code that handles the loading of the terrain data and other propagation information and calling the existing ITWOM or Langley Rice Code. In addition to being more maintainable and being structured to allow easier parallelization, such a program might have the following new features:

- Support for external elevation databases. The USGS, Google and others maintain web services that provides elevation data from its database for a user given area. This would save the user from having to maintain a local elevation database.

- Support for other GIS information for clutter and other properties. Currently, SPLAT! assumes a uniform amount of "clutter" above the ground throughout the mapped area. With data sources like the Openstreetmap project and other geographical databases, it should be possible to estimate the real height of "clutter" on a path-by-path basis. This might improve the accuracy of propagation estimates in an area.

- Support for dynamically loading terrain and elevation data. SPLAT! currently loads ALL elevation data before running a single propagation calculation. Since each path is independent with Langley Rice, terrain can be loaded dynamically and discarded when no longer needed. This could reduce the program's memory footprint over large areas and it might even improve performance by allowing propagation calculations to occur while other pieces of terrain are being loaded.

A Complete Source Code and Data

The complete source code, revision history and data used in the project are available at http://github.com/amusselm/Parallel-LRP

References

[1] Alex Farrent. Discussion of SPLAT and CloudRF via email. 20 July 2013.

[2] Federal Communications Comission. *In the Matter of Establishment of a Model for Predicting Digital Broadcast Television Field Strength Received at Individual Locations. Measurement Standards for Digital Television Signals Pursuant to the Satellite Home Viewer Extension*, Proceeding 10-152, 21 January 2011.

[3] Free Software Foundation. GNU General Public License. World Wide Web, 2 June 1991.

[4] Benedict Gaster, Lee Howes, David R. Kaeli, Perhaad Mistry, and Dana Schaa. *Heterogeneous Computing with OpenCL*. Morgan Kaufmann, 2011.

[5] Intel Corperation. Introducing the Intel(R) SDK for OpenCL* - Debugger for Linux* OS. World Wide Web, 19 December 2012.

[6] Intel Corperation. Intel Ark - Intel Xeon Processor E5504 (4M Cache, 2.00 GHz, 4.80 GT/s Intel QPI). World Wide Web, 29 April 2013.

[7] John A. Magliacane. Splat Website. World Wide Web, 14 March 2011.

[8] Khronos Group. OpenCL - The open standard for parallel programming of heterogeneous systems. World Wide Web, 25 March 2013.

[9] A.G. Longley and P.L. Rice. Prediction of tropospheric radio transmission loss over irregular terrain. a computer method-1968. Technical report, DTIC Document, 1968.

[10] S.E. Shumate. Longley-rice and itu-p.1546 combined: A new international terrain-specific propagation model. In *Vehicular Technology Conference Fall (VTC 2010-Fall), 2010 IEEE 72nd*, pages 1 –5, sept. 2010.

[11] Yang Song and A. Akoglu. Parallel implementation of the irregular terrain model (itm) for radio transmission loss prediction using gpu and cell be processors. *Parallel and Distributed Systems, IEEE Transactions on*, 22(8):1276 –1283, aug. 2011.

[12] US Department of Commerce. Irregular Terrain Model (ITM) (Longley-Rice). website. US Department of Commerce National Telecommunications and Information Administration Institute for Telecommunication Sciences.