Real-Time Radio Wave Propagation for Mobile Ad-Hoc Network Emulation Using GPGPUs

Brian J. Henz*, David Richie†, Evens Jean‡,Song Jun Park*, James A. Ross§, Dale R. Shires*
*U.S. Army Research Laboratory, APG, MD 21005
†Brown Deer Technology, Forest Hill, MD 21050
‡Secure Mission Solutions, APG, MD 21005
§Dynamics Research Corporation, Reston, VA 20190

***Abstract*—The accurate simulation and emulation of mobile radios requires the computation of RF propagation path loss in order to accurately predict connectivity and signal interference. There are many algorithms available for computing the RF propagation path loss between wireless devices including the Longley-Rice model, the transmission line matrix (TLM), ray-tracing, and the parabolic equation method. Each of these methods has advantages and disadvantages but all require a large number of floating point operations during execution. In this paper we investigate using general purpose graphics processing units (GPGPUs) to provide the computational capabilities required to perform these RF path loss calculations in real-time in order to support large scale mobile ad-hoc network emulation. Three specific methods, namely the Longley-Rice, TLM, and ray-tracing methods are explored including usage cases and performance analysis on GPUs. The Longley-Rice algorithm is solved in real-time for 1000's of transmitters and receivers, the TLM method is well suited for GPU acceleration as is ray-tracing. We will discuss the algorithm modifications required for efficient GPU use, precision issues and optimization.**

Keywords: Mobile Ad-Hoc Network, Emulation, GPGPU, RF Propagation Path loss, Longley-Rice, Transmission Line Matrix, Ray-Tracing

I. Introduction

Large scale testing, evaluation and analysis of mobile ad-hoc network (MANET) platforms is an expensive proposal with a limited parameter space and repeatability under experimental conditions. [1] Therefore, simulation and emulation tools have been developed that provide researchers with a controllable and repeatable environment for analysis of MANET platforms. In particular, emulation holds great promise for limiting the amount of live experimentation required for MANET platform development. Emulation provides for hardware-in-the-loop (HIL) testing and analysis where the physical medium is replaced by a virtual environment and a physical or simulated radio can be used with real applications. Much effort has been performed in this area to make the virtual environment as physically meaningful as possible [1]–[3] but one limitation that remains for real-time emulation is the accurate computation of the RF propagation path loss between radios.

RF propagation path loss predictions for MANET emulation has traditionally relied on either off-line link analysis using various models including high fidelity finite difference time domain (FDTD) and ray-tracing methods [2] or real-time calculations with stochastic models. [3] When calculations are performed off-line it is assumed that either the node mobility is known apriori or some large data set of node locations are computed and stored in a look-up table. Limiting the mobility apriori can be a severe limitation when experiments may involve live components or mobility is controlled by a third party application such as a force modeling simulation. One method to remove this limitation is to use interpolation between known data points but the accuracy and efficiency of this method is limited by a number of factors. These factors include the physical size of the virtual environment, machine memory for storing and accessing a look up table, signal phase, and fading affects from small obstructions are not captured because of the computed grid size. Computationally inexpensive methods such as the various free-space models are not a satisfactory solution either, as they do not capture the effect of terrain, vegetation, precipitation or man-made structures on RF propagation path loss.

RF propagation models play an essential role in the planning, analysis and optimization of radio networks [1], [4]–[7]. For instance, coverage and interference estimates of network configurations are based on field strength predictions, routing is also highly dependent upon computed path loss data. [1] The increasing fidelity of MANET emulations from packet-level to signal-level [8] analysis will require fast and accurate modeling of the physical layer. [9], [10] Using GPUs to provide the floating point performance required to compute the RF propagation path loss algorithms in real time it is possible to provide a more realistic physical layer for MANET emulations and simulations. The first algorithm discussed will be the Longley-Rice method as implemented within the irregular terrain model (ITM). The ITM is well suited for large scale emulations of 1000's of devices located in a non-urban environment. The second method investigated is the transmission line matrix or TLM which is targeted towards pico-cell scenarios within buildings or in relatively localized urban environments. The final method investigated, the ray tracing method, is used primarily for small scale to large scale urban environments.

II. Background

The scale and complexity of MANETs used by the Department of Defense (DoD) continues to increase, and is increasing particularly within the Army as a mobile fighting force. The

military is rapidly becoming a network-centric force, with substantial access to sensor-derived surveillance information as well as an increasingly complicated application layer running over many different devices. Each layer introduces significant advantages to the war fighter, but also brings in new dependencies and new risks from the rapid change in configurations of the MANETs that provide network access across the battlefield. Headed by the U.S. Army Research Laboratory (ARL), the Mobile Network Modeling Institute (MNMI) was established in 2007 to exploit High Performance Computing (HPC) resources through the development of computational software. Thus enabling the DoD to design, test, and optimize networks at sufficient levels of fidelity and with sufficient speed to understand the behavior of network technologies in the full range of conditions under which they will be deployed. Operational goals of the MNMI include the development of scalable computational modeling tools for simulations and emulations, the ability to understand apriori the performance of proposed radio waveforms in the field, and to optimize the network for U.S. Army war fighters. The results of the MANET modeling effort presented here are from an effort at the ARL that is focused on the development of a framework for large scale MANET emulations, e.g. up to 5000 emulated devices. A large scale emulation environment will provide a testbed for the research, development, and evaluation of network algorithms, applications and devices in a controlled environment. [11]

III. RF Propagation Path Loss Algorithms

As previously mentioned, there are many approaches for field strength prediction and they can be roughly divided into semi-empirical, time-domain methods and ray-optical models. For example, the semi-empirical COST-Walfisch-Ikegami model [12] estimates the received power predominantly on the basis of frequency and distance to the transmitter. Ray-optical [13] approaches identify ray paths through the scene, based on wave guiding effects like reflection and diffraction. Semi-empirical algorithms usually offer fast computation times but suffer from inherent low prediction quality. Ray-optical algorithms feature a higher prediction quality at the cost of higher computation times, while time-domain methods typically increase accuracy further with even higher computational costs.

At the physical layer, the interactions between devices is governed by the RF propagation characteristics of the environment. MANET emulation with HIL capabilities further require that the RF path loss data must be computed and provided to the emulation environment in real-time. The algorithms used to compute path loss must be computed in real-time for each of the possible propagation paths. Initially assuming that all devices in a single emulation scenario are within propagation range of each other the computational complexity of the RF path loss algorithm is $O\left(n^2\right)$, where n is the number of transmitter/receiver device pairs in the scenario. Although the computational cost varies, the methods available for computing the RF path loss data all require a large number of floating point operations, necessitating a high FLOP (floating point operations) rate for real-time path loss predictions.

Recently, the use of GPUs has been identified as a solution to provide the raw floating point performance [14] required to compute the RF propagation path loss in real-time. [6], [7] Originally, GPUs were developed in order to quickly compute rasterization which requires a large number of simple floating point operations. This targeted design is the reason that the architecture has been able to exceed the performance of CPU architectures for raw FLOP rates. [15] The MANET emulation environment used here is EMANE (Extendable Mobile Ad-hoc Network Emulator) from DRS (formerly Cengen Labs) [16]. In EMANE, the GPS locations of all mobile radios are transmitted over an IP multicast group that is monitored by the emulated devices for self-location.

Although a number of path loss algorithms exist, we down-selected the methods based on various scenarios we typically encounter. For instance we have criteria for large scale non-urban environment, large scale urban environments and very localized analysis for moderate numbers of devices. In order to provide a robust path loss calculation for the non-urban environments we selected the Longley-Rice model. The Longley-Rice model is capable of predicting path loss in an area or point to point mode, with the later used here. Longley-Rice is designed for frequencies between 20 MHz and 20 GHz and for path lengths between 1 km and 2000 km [17], both within our scenario operating ranges. In point-to-point mode the model considers input parameters such as distance, antenna height, surface reflectivity, climate and the terrain profile between the transmitter and receiver. [18] The rest of the environmental parameters can be transient or fixed upon initialization. This implementation is robust in that it allows all parameters to change each time the GPU kernel is executed. The TLM method [19] is related to the FDTD method and as such discretizes space and time for computing the electromagnetic field. One advantage of the TLM method over FDTD for this application is the larger spatial discretization possible, and is well-suited for analysis of local areas or pico-cells. The final method investigated is ray tracing [13], using the shooting bouncing ray (SBR) method. The ray tracing method is computationally expensive but many of the algorithms required to compute this method translate efficiently onto GPUs, and is capable of producing results for large urban environments. In the interest of space we will give details below on the implmentation of the TLM algorithm before discussing the achieved performance for all three methods.

A. The Transmission Line Matrix Algorithm for Real-Time Radio Wave Propagation Path Loss in Pico-Cells

The transmission line matrix method or transmission line modeling method relies on the relationship between electromagnetic field quantities and voltage and current on transmission lines. [20] The formulation followed in this work is the three-dimensional symmetrical condensed node (SCN). [21] Although this method is more efficient that the FDTD method

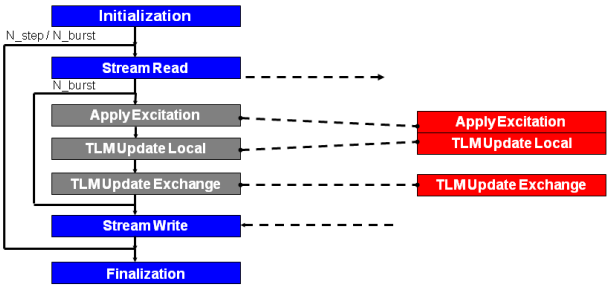

Fig. 1. TLM kernels showing those ported to GPU on the right and where communication can be limited but increasing the number of steps per burst.

that requires approximately 10 grid points per wavelength, each node in the cubic mesh requires solution of 12 values. More details of this algorithm can be found elsewhere [19]–[22].

1) TLM Implementation Details for the GPU: The TLM implementation used in this work is solved using a regular Cartesian grid and therefore memory accesses from neighboring grid points is well defined and memory efficient. This is very important for porting of an application to the GPU architecture as cache misses are much more expensive than on a general purpose CPU. In our implementation the memory access is based on a 3D stencil and calculations are mostly MADDs (Multiply-Add). The TLM code is based on a previous FDTD code optimized for GPUs using the Brook+ language. The algorithm, illustrated in Figure 1, is composed of 7 primary functions. Of these functions Initialization, Stream Read, Stream Write and Finalization are executed on the CPU. The functions within the N_burst loop, namely Apply Excitation, TLM Update Local and TLM Update Exchange are all executed on the GPU hardware. By increasing the ratio of computation to communication the transfer of data across the PCIe bus between CPU and GPU can be limited, potentially increasing performance significantly.

In the TLM implementation used here the number of time steps (N_step) for which the entire grid of Voltages is computed is predefined. This is reasonable since for a regular grid size, the maximum distance that a wave will travel before the input Voltage is insignificant can be estimated from the medium attenuation coefficient. The maximum mesh size, $\triangle l$, can be estimated from the following equation.

$$\frac{\triangle l}{\lambda} \leq 0.1 \tag{1}$$

Where $\triangle l$ is the mesh size and λ is the wavelength of interest. [20]

2) TLM algorithm optimization for GPUs: As previously discussed, the TLM method is well suited for the GPU architecture. An important optimization developed by one of the authors is called shuffled grids. Using this method it is possible to efficiently combine 4 single precision floating point operations of the TLM method into a single float4 SIMD

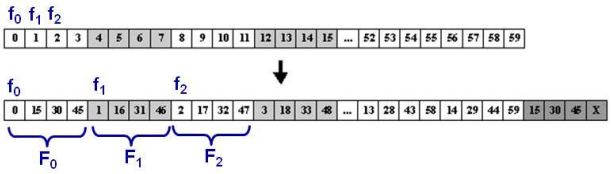

Fig. 2. The memory layout for the shuffled grid method using a 15 offset.

operation. In Figure 2 consider the simple 1D stencil.

$$g_1 = a * f_0 + b * f_1 + c * f_2 \tag{2}$$

In order to leverage float4 SIMD operations the memory is shuffled as shown in Figure 2. The calculation from Equation 2 is then rewritten using float4 SIMD operations.

$$G_1 = c_0 * F_0 + c_1 * F_1 + c_2 * F_2 \tag{3}$$

Which is equivalent to performing the following set of calculations.

$$g_1 = a * f_0 + b * f_1 + c * f_2 \tag{4a}$$
$$g_{16} = a * f_{15} + b * f_{16} + c * f_{17} \tag{4b}$$
$$g_{31} = a * f_{30} + b * f_{31} + c * f_{32} \tag{4c}$$
$$g_{46} = a * f_{45} + b * f_{46} + c * f_{47} \tag{4d}$$

Although there is a small amount of bookkeeping associated with the shuffling and unshuffling of grid points, these are performed as pre- and post-processing steps with little overhead. While the potential performance gains for the TLM algorithm are close to a 4x speedup.

IV. Numerical Stability and Consistency Across Architectures

As noted previously, single precision floating point computations are used on the GPU in order to achieve maximum performance, the trade-off being possibly decreased accuracy. Since the Longley-Rice model uses statistical estimates to compute the variability of signal path loss due to situation, time and location. The actual received signal is expected to deviate from the computed value due to these variables but the model still provides a reasonable estimate. Therefore, small variations due to single versus double precision are not expected to invalidate the computed results for its intended purpose of estimating signal loss over irregular terrain. For the Longley-Rice algorithm there are a large number of transcendental functions that do not have a double precision computation available. Algorithm development with single-precision accuracy raised concerns with numerical stability and consistency, especially, in the context of forward and inverse transcendental functions with small angles. Whereas it is possible, although not guaranteed, that reasonably precise consistency might be expected across these architectures for simple algorithms based on multiply-add operations, the complexity and reliance upon complex transcendental operations makes exact agreement here unlikely. Factors impacting the

difference in results include extended bit precision used in some operations, differences in rounding behavior, and differences in the software implementation of complex operations. Additionally, the GPU implementation introduces the possibility of order-of-operation effects as a result of the fine-grain parallelism within some kernels.

An issue identified across many elements of the algorithm was the repeated use of forward and inverse transcendentals at small angles. An example of this small-angle effect is the use of great circle calculations over small areas in which the correction due to the curvature of the earth was small. A serious numerical instability was identified with the pattern of successive operations of cosine, followed by a minor calculation, and then followed by an arc cosine. Such patterns had the potential to produce an intermediate value slightly greater than 1.0 and a final result of NaN (not a number). The effects of this numerical instability can be complicated and the impact on the final path loss can range from a small error to an undefined result (NaN). In some cases a less severe numerical error results from differences in transcendental functions at limiting values. Secondary impacts were also identified, for example differences in the projected map location within the digital terrain map can introduce differences in elevation within the extracted height profile that only impact results by changing the statistical metrics calculated for these height profiles. The solution to many of these issues was to re factor the formulas found in the original reference implementation and introduce forms with greater stability at the limiting ranges found within the typical uses cases. Consider the original distance calculation, that begin by first calculating,

$$\begin{aligned} a = &\cos(90 - lat_2) * \cos(90 - lat_1) + \\ &\sin(90 - lat_2) * \sin(90 - lat_1) * \\ &\cos(lon_2 - lon_1) \end{aligned} \tag{5}$$

Where lat_1, lon_1 refer to the transmitter coordinates and lat_2, lon_2 refer to the receiver coordinates. Using the value a computed in Equation (5),

$$b = \arccos(a) \tag{6}$$

Where for the earth,

$$distance = R_{earth} * b \tag{7}$$

Here R_{earth} is the radius of the earth. For small angles this calculation can be unstable using single precision so we used the following approximation,

$$\Delta lon = lon_2 - lon_1 \tag{8a}$$

$$\Delta lat = lat_2 - lat_1 \tag{8b}$$

$$\begin{aligned} a = &(\sin(\Delta lat))^2 + \\ &\cos(lat_1) * \cos(lat_2) * \left(\sin\left(\frac{\Delta lon}{2}\right)\right)^2 \end{aligned} \tag{9}$$

$$b = 2 * \arcsin(\min(1, \sqrt{a})) \tag{10}$$

Distance is then computed using Equation (7). Efforts to improve the numerical stability resulted in good agreement between a CPU and AMD Cypress and Cayman GPUs. We take as an assumption that the CPU hardware provides a reasonable baseline for comparison since the implementation of all relevant math operations are well established, more thoroughly tested, and provide better edge cases relative to GPUs. Results for the NVIDIA Fermi GPU exhibited notable discrepancies, with a complete understanding of the cause remaining for further investigation. Numerical consistency was tested across these architectures using a simple synthetic test case involving an 8 by 8 uniform grid of radio transceivers over a DEM (digital elevation map) with 1.2M elevation points. Table I shows the percentage of the point-to-point path loss results calculated on a particular GPU architecture that agree with the results calculated on the CPU to within a tolerance of 1 dB, 2 dB, and 10 dB, respectively.

TABLE I
CONSISTENCY OF THE RESULTS CALCULATED WITH VARIOUS GPUS COMPARED TO THE BASELINE RESULTS FROM THE CPU.

Processor	<1 dB	<2 dB	<10 dB
ATI Radeon HD 5870	98 %	99 %	100 %
AMD Radeon HD 6970	98 %	99 %	100 %
NVIDIA Tesla C2070	86 %	90 %	94 %

As observed in Table I, the ATI/AMD devices provide a result more consistent with the baseline CPU. We have been unable to determine at this time the cause of the discrepancy between the two vendors but the ATI/AMD solution consistently provided results more consistent with the CPU baseline calculations.

V. PERFORMANCE AND SCALING

In this section we explore the achieved performance on each of the algorithms on several GPU platforms. In the process comparing vendor we also compare solutions from ATI/AMD and NVIDIA. Each of the algorithms has its own peculiarities that affect performance, for instance the Longley-Rice algorithm is heavily dependent upon transcendental functions and not on more typical MADD (multiply add) operations, whereas TLM has very structured memory accesses and contains almost exclusively MADD operations. This results in come interesting comparisons as the reported FLOP rates are for MADD operations, and transcendental function performance is not directly related.

A. ITM Performance

The ITM algorithm was the first method investigated and therefore this section contains a number of results and comparisons. We start by giving the overall application computation times in Table II which lists the wall clock time required for three different architectures to compute all point-to-point RF path loss values using the Longley-Rice algorithm.

As illustrated in Table II using the current ITM implementation, all of the tested GPU architectures are capable of

TABLE II
TIMING RESULTS FOR 256 TRANSMITTERS/RECEIVERS USING THE OPENCL VERSION OF THE LONGLEY-RICE ALGORITHM RUN ON AMD AND NVIDIA GPUS.

Processor	Time (s)
ATI Radeon HD 5870	0.72
AMD Radeon HD 6970	0.55
NVIDIA Tesla C2070	0.39

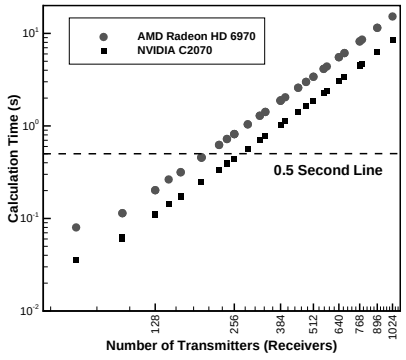

Fig. 3. Plot of total ITM (Longley-Rice) calculation time versus number of transmitters/receivers. The 0.5 second line represents the maximum time allowed for real-time computations.

providing computed RF path loss results for 256 transceivers, or 65,536 point-to-point calculations, in less than 1 second on a single GPU device. For 256 radios, the fastest time to solution is reported as 0.39 sec using an NVIDIA C2070 [23] as compared with 0.72 sec and 0.55 sec using an ATI Radeon HD 5870 and AMD Radeon HD 6970, respectively. [24] Complete performance results are plotted in Fig. 3 for a range of 32 to 1024 transceivers.

In Figure 3 a line is drawn at 0.5 seconds to show approximately the number of transceivers a particular GPU is capable of considering in real-time. It is interesting to note that the theoretical peak FLOP rate of the AMD Radeon HD 6970 is 2.703 TFLOPs and the NVIDIA C2070 is only 1.288 TFLOPs. Conversely, the number of radios supported by the Longley-Rice algorithm in less than 0.5 seconds of computation time is higher for the NVIDIA GPU. This apparent inefficiency in the AMD hardware is due to the fact that many of the floating-point operations in the Longley-Rice algorithm are transcendental functions such as cosine, sine, tangent, co secant, etc., The performance of a specific architecture on the Longley-Rice algorithm is therefore not easily predicted by theoretical peak performance. Additionally, the memory access patterns within the kernels are non-trivial, and this will also contribute to the observed performance.

Performance of complex multi-kernel algorithms can be impacted by many factors including pure computational load, memory access, host-device data transfer, and kernel launch latency. In the case of the 10 kernels in the ITM implementation, each individual kernel shows a very low execution time when directly measured in a fully blocking mode of operation. In order to investigate whether the ITM implementation is effectively using the GPU compute capability, the stripe size over which the computation is distributed was varied to observe the effect of changing the amount of work performed per kernel execution. Initially the stripe size was set at 4096 with subsequent test cases of 2048 and 1024 point-to-point calculations. The results in Table III show an improvement on the order of 10% when increasing the block size from 1024 to 4096, thus providing more work per kernel execution. This indicates that the block size of 4096 is performing only slightly better than the block size of 2048, therefore increasing the block size further would yield diminishing returns. Increasing the block size further would also decrease the efficiency of performing calculations where the number of point-to-point paths was not commensurate with block size. For example, with a workload of 65536 point-to-point calculations, increasing the block size will approach the size of the work load resulting in an efficient calculation when the work load is not a multiple of the block size.

TABLE III
PERFORMANCE FOR AMD AND NVIDIA GPUS AS A FUNCTION OF BLOCK SIZE IN TERMS OF THE NUMBER OF POINT-TO-POINT PATHS EVALUATED PER KERNEL EXECUTION.

Processor	Block Size	Time (s)
ATI Radeon HD 5870	1024	0.83
ATI Radeon HD 5870	2048	0.75
ATI Radeon HD 5870	4096	0.72
AMD Radeon HD 6970	1024	0.65
AMD Radeon HD 6970	2048	0.58
AMD Radeon HD 6970	4096	0.55
NVIDIA Tesla C2070	1024	0.42
NVIDIA Tesla C2070	2048	0.40
NVIDIA Tesla C2070	4096	0.39

B. TLM Performance

As noted previously, the TLM algorithm, much like FDTD, is well suited for the GPU architecture. In this case the biggest bottleneck is expected to be data transfer across the PCIe bus which is known to be a bottle neck for applications executing on GPUs. By limiting the number of times results are transported across the PCIe bus in the TLM algorithm we were able to optimize the calculation time by an order of magnitude, Figure 4.

Notice in Figure 4 that the time per step for CPUs remains fairly constant from 10 to 1000 steps, whereas the GPU results show an order of magnitude decrease in time per step. This illustrates the importance of increasing the computation to communication ratio when using GPUs as a co-processor. In Figure 4 the cpu-opt and gpu-opt lines refer to the use of the shuffled grid method discussed previously. The gpu-opt time per step line shows a nearly ideal 4x speedup over the unoptimized version, whereas the cpu-opt line shows

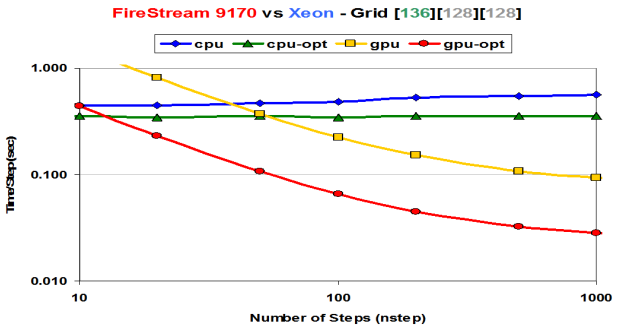

Fig. 4. Plot of time per step computed showing performance increases for GPUs with modest gains for CPUs. By increasing the nsteps parameter the ratio of computation to communication is increased. Notice the power scale on the y-axis.

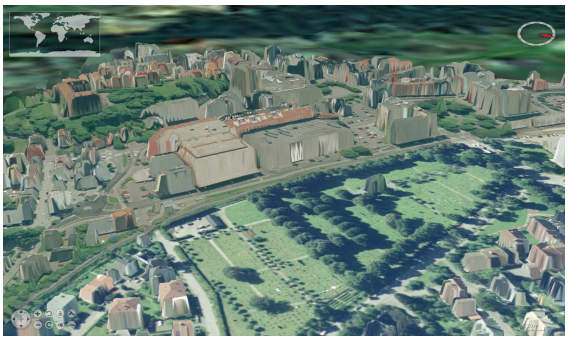

Fig. 5. Side view of 3D polygon data from Tonsberg, Norway used for ray tracing algorithm development.

about a 1.2x speedup over the unoptimized CPU version. The nearly 4x speedup indicates that the algorithm is able to take advantage of the GPU ability to perform MADD functions on 4 32-bit floating point values simultaneously.

C. Ray Tracing Performance

The primary factor in determining execution time for the ray tracing algorithm is the number of rays generated and computed. The total number of rays in the system depends on the number of rays emitted by individual transmitters, the number of reflections, scattering, diffraction and refraction allowed as well as the number of planar surfaces with which the rays can interact. Note that in this experiment, we are solely focusing on emitted and reflected rays. The environment in use in our research is a polygon-based 3D representation of the town of Tonsberg, Norway, Figure 5. As noted earlier this model contains 68,356 triangle and the benchmark scenario contains two transmitters that spherically emit rays in all directions. The emission angles of individuals rays is dependent upon the user-specified n_θ and n_ϕ values. The values of n_θ and n_ϕ are equivalent over each run and vary their values between 64, 128 and 256. Each emitted ray is traced throughout the environment to generate reflected rays based on their interactions with the planar surfaces. The path of the reflected ray is computed based on the laws of reflection in light propagation.

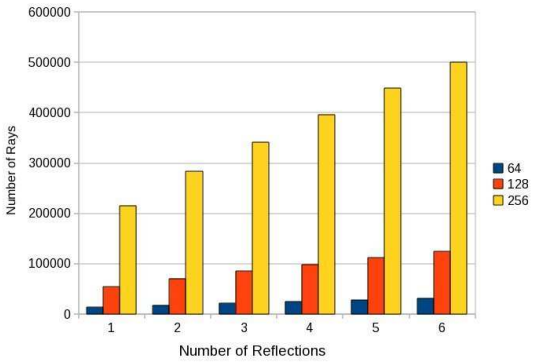

Fig. 6. The number of rays generated during the ray tracing calculation with a maximum of 1 to 6 reflections and an angular partitioning of 64, 128 or 256 partitions.

The ray tracing method is developed using OpenCL in order to take advantage of multiple platforms, although for these results we used an NVIDIA Quadro FX4800 GPGPU with 1.5GB GDDR3 of GPU memory. The total number of rays in the system is computed, and Figure 6 plots the number of rays in the system for n_θ and n_ϕ as they vary from 64 to 256. In Figure 6 the maximum number of reflections that individual rays are permitted to undergo is varied from 1 to 6. It is expected that some predetermined maximum number of reflections or unfolded ray length will be used to limit the run time while preserving accuracy.

Using the parameters from Figure 6 the run time for each configuration is collected and plotted against the number of rays generated, Figure 7. Figure 7 shows a linear relationship between run time and number of rays, but with offsets depending on the initial angular partitioning used. This is related to the cost of initial ray generation and generating new rays after intersection with a surface has occurred. The offsets are approximately equivalent to the difference between the squares of the number of angular partitions. E.g. $128^2 - 64^2 = 12288$ and $256^2 - 128^2 = 49152$. Note that for each ray, the system currently needs to analyze all of the 68,356 planar surfaces in the environment to determine its endpoint. Future performance enhancements will therefore focus on the size of the model and the number of polygons that need to be interrogated for each ray.

VI. Conclusions and Future Work

MANET emulation of large scale networks is a useful tool for network analysts but without realistic RF propagation the accuracy of the results are questionable. Using GPUs we have developed three RF propagation path loss methods that can run in real time or near real time along side a MANET emulation to provide realistic path loss data. These algorithms cover a broad range of the typical scenarios encountered by MANETs in the field, namely, non-urban large networks, large scale urban networks and pico-cells of around 20 nodes in a local area. We have investigated the use of the standard OpenCL language against vendor solutions such as Brook+ and CUDA.

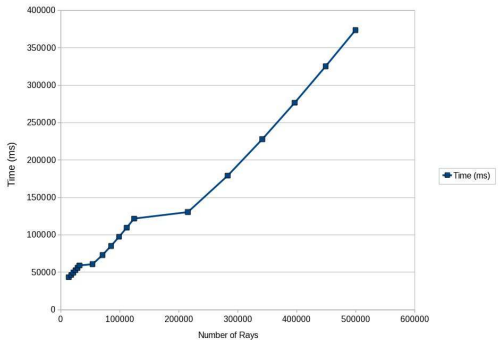

Fig. 7. Ray Tracing algorithm run time versus number of generated rays. The three different slopes correspond to the number of initial angular partitioning of 64, 128 and 256 partitions.

We have also shown how algorithm development for GPUs is very important for achieving maximum performance, such as the shuffled grid method, modifying calculations to use single precision where possible, etc. Additionally for the ITM it was possible to use reduced precision calculations, through the use of alternative calculations for edge cases to improve performance on GPUs. Load distribution and communication costs were mimed by the creation of computation blocks that limit kernel calls and minimize wasted computation cycles. These developments enable the emulation framework at ARL to provide real time situational awareness data to live field exercises and will have applicability to the integration with future modeling simulations and the fielding of upcoming devices.

Although vendor supplied languages for GPUs have shown to currently provide superior performance we have settled on using a portable standard for parallel computing systems, namely OpenCL. Using OpenCL we have developed the Longley-Rice ITM and ray tracing methods for real time RF path loss computations that supports MANET emulation. Enabled MANET emulation provides the capability to augment live exercises, integrate MANET emulation with simulations and to drive programmable attenuators for laboratory experimentation with physical devices. Prior to the development of these capabilities with GPUs, the wireless node mobility and path loss for a scenario needed to either be computed apriori or to use a large number of (i.e. 10,000) CPU cores, dedicated to path loss calculation. This was not acceptable because the CPUs cores are required to host virtual machines for MANET emulation and by using GPU co-processors it has been possible to over come this hurdle for efficient large scale MANET emulation.

VII. Acknowledgments

The authors would like to acknowledge the support received from the High Performance Computing Modernization Program Office (HPCMPO) under the Mobile Network Modeling Institute (MNMI). The authors would also like to acknowledge the support of the HPCMP PETTT (High Performance Computing Modernization Program Productivity Enhancement, Technology Transfer and Training) program.

References

[1] Arne Schmitz and Martin Wenig. The effect of the radio wave propagation model in mobile ad hoc networks. In *The 9th ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM 2006)*, October 2006.

[2] Michael A. Kaplan, Ta Chen, Mariusz A. Fecko, Provin Gurung, Ibrahim Hokelek, Sunil Samtani, Larry Wong, Mitesh Patel, Aristides Staikos, and Ben Greear. Realistic wireless emulation for performance evaluation of tactical manet protocols. In *IEEE Military Communications Conference (MILCOM)*, October 2009.

[3] Thomas Nitsche and Thomas Fuhrmann. A tool for raytracing based radio channel simulation. In *SIMUTools*, March 2010.

[4] T. Rick and R. Mathar. Fast edge-diffraction-based radio wave propagation model for graphics hardware. In *Antennas, 2007. INICA '07. 2nd International ITG Conference on*, pages 15–19, March 2007.

[5] Glenn Judd and Peter Steenkiste. Design and implementation of an rf front end for physical layer wireless network emulation. In *IEEE 65th Vehicular Technology Conference (VTC2007)*, April 2007.

[6] D. Catrein, M. Reyer, and T. Rick. Accelerating radio wave propagation predictions by implementation on graphics hardware. In *Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th*, pages 510 –514, april 2007.

[7] David Michéa and Dimitri Komatitsch. Accelerating a three-dimensional finite-difference wave propagation code using gpu graphics cards. *Geophysical Journal International*, 182(1):389–402, 2010.

[8] P. Andelfinger, J. Mittag, and Hartenstein. H. GPU-based Architectures and their Benefit for Accurate and Efficient Wireless Network Simulations. In *IEEE MASCOTS*, pages 421–424, July 2011.

[9] I.K. Eltahir. The impact of different radio propagation models for mobile ad hoc networks (manet) in urban area environment. In *Wireless Broadband and Ultra Wideband Communications, 2007. AusWireless 2007. The 2nd International Conference on*, page 30, aug. 2007.

[10] Illya Stepanov and Kurt Rothermel. On the impact of a more realistic physical layer on manet simulations results. *Ad Hoc Networks*, 6(1):61 – 78, 2008.

[11] Esten Ingar Grøtli and Tor Arne Johansen. Path planning for uavs under communication constraints using splat! and milp. *J. Intell. Robotics Syst.*, 65(1-4):265–282, January 2012.

[12] E. Damasso, editor. *Digital mobile radio towards future generation systems*. Office for Official Publications of the European Communities, Luxembourg, 1999.

[13] Henry L. Bertoni. *Radio Propagation for Modern Wireless Systems*. Prentice Hall Professional Technical Reference, 1999.

[14] Yang Song and Ali Akoglu. Parallel implementation of the irregular terrain model (itm) for radio transmission loss prediction using gpu and cell be processors. *IEEE Trans. Parallel Distrib. Syst.*, 22:1276–1283, August 2011.

[15] NVIDIA. *NVIDIA CUDA C Programming Guide, Version 4.0*, 2011.

[16] Various. *EMANE Developer Manual 0.7.3*. DRS CenGen, LLC, 2012.

[17] G.A. Hufford, A.G. Longley, and W.A. Kissick. A Guide to the Use of the ITS Irregular Terrain Model in the Area Prediction Mode. Technical Report 82-100, National Telecommunications and Information Administration, April 1982.

[18] G. Hufford. *The ITS Irregular Terrain Model, version 1.2.2 The Algorithm*. National Telecommunications and Information Administration Institute for Telecommunication Sciences, 1995.

[19] C. Christopoulos. *The Transmission Line Modeling Method: TLM*. IEEE Press, 1995.

[20] Sadasiva M. Rao. *Time Domain Electromagnetics*. Academic Press, 1999.

[21] P.B. Johns. A symmetrical condensed node for the tlm method. *Microwave Theory and Techniques, IEEE Transactions on*, 35(4):370 – 377, apr 1987.

[22] P. Naylor and R. A. Desai. New Three Dimensional Symmetrical Condensed Lossy Node for the Solution of Electromagnetic Wave Problems by TLM. *Electronics Letters*, 26(7):492–494, 1990.

[23] Benchmarks used the OpenCL implementation provided by the NVIDIA CUDA Toolkit v3.2.

[24] Benchmarks used the OpenCL implementation provided by the AMD ATI Stream SDK v2.3.