Ultra-High Fidelity Radio Frequency Propagation Modeling Using Distributed High Performance Graphical Processing Units

A simulator for multi-element non-stationary antenna systems

Mark Barnell
Air Force Research Lab (AFRL)
Information Directorate
Rome, NY 13441 USA
mark.barnell.1@us.af.mil

Nathan Stokes, Jason Steeger, Jessie Grabowski
SRC, Inc
North Syracuse, NY 13212 USA
stokes@srcinc.com
jsteeger@srcinc.com

***Abstract*— A newly-invented, distributed, high-performance graphical processing framework that simulates complex radio frequency (RF) propagation has been developed and demonstrated. The approach uses an advanced computer architecture and intensive multi-core system to enable high-performance data analysis at the fidelity necessary to design and develop modern sensor systems. This widely applicable simulation and modeling technology aids in the design and development of state-of-the-art systems with complex waveforms and more advanced downstream exploitation techniques, e.g., systems with arbitrary RF waveforms, higher RF bandwidths and increasing resolution. The recent breakthroughs in computing hardware, software, systems and applications has enabled these concepts to be tested and demonstrated in a large variety of environments and early in the design cycle. Improvements in simulation accuracies and simulation timescales have been made that immediately increase the value to the end user. A near-analytic RF propagation model increased the computational need by orders of magnitude. This model also increased required numerical precision. The new general purpose graphics processing units (GPGPUs) provided the capability to simulate the propagation effects and model it with the necessary information dependence, and floating point mathematics where performance matters. The relative performance improvement between the baseline MATLAB® parallelized simulation and the equivalent GPU based simulation using 12 NVIDIA Tesla K20m GPUs on the Offspring High-Performance Computer (HPC) using the AirWASP© framework decreased simulation and modeling from 16.5 days to less than 1 day.**

Keywords— Distributed computing; general purpose GPU computing; RF Propagation Modeling; multi-element antenna system modeling)

I. INTRODUCTION

Radio frequency (RF) propagation simulation is an important aspect of any modern system that uses an RF link, as it enables the system to be tested in a large variety of environments early in the design cycle. System design factors such as wide signal bandwidths, gigahertz and higher frequencies and multiple input / multiple output (MIMO) antenna architectures and adaptive interference rejection necessitate a simulation that is nearly analytic in fidelity. Furthermore, simulating a system where distance is measured in kilometers and propagation delay is measured in fractions of a nanosecond requires double-precision floating-point computations. While modern Intel Xeon processors offer significant floating-point operations per second (FLOPS), they are still several orders of magnitude less than the equivalent generation GPGPU [4].

Fig. 1 illustrates the computational requirements of the propagation model for a single element for one second of simulated time. These values are for a 50 MHz sampling rate and are shown as a function of scatterers modeled. The problem quickly becomes exa-scale for multi-element systems with even moderate numbers of scatterers modeled.

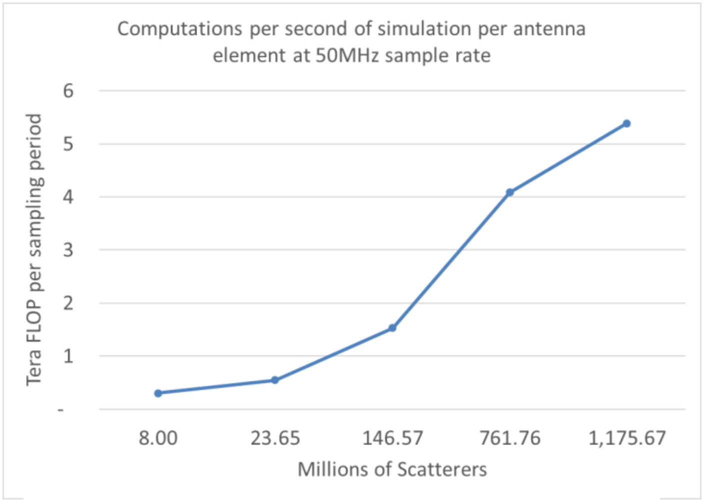

Fig. 1. Example Computational Requirements

For instance, to model the AN-SPY1 RADAR which has 4096 transmit elements and 4352 receive elements per face, would require 9.8e18 FLOP per second for the 8 million scatterer case[5].

II. Background / significance

Quantifying the effects of environmental scatterers on a propagated RF signal is an important aspect of modern RF system design given the requirements for lower size, weight and power (SWaP), coexistence in crowded RF spectrums, and performance in large urban areas. Modeling the effects can help system engineers understand the clutter response in a RADAR system or the interference rejection capabilities of a communications system.

SRC, Inc developed a new ultra-high fidelity RF propagation simulation to support design validation and analysis of state-of-the-art RF systems. To effectively model a discretely sampled RF system that has a wideband signal and multiple antenna elements the actual propagation path between the emitting element, the scattering element and the receive element must be computed independently. Furthermore, to capture the effects of a wideband signal, the signal cannot be approximated as an ideal impulse response, and must be accurately modeled for each sample received. It should be noted, this simulation does not model multi-path, which is an entirely separate exercise in ray-tracing.

Fig. 2 and *Fig. 3* show the the simulation artifacts in angle Doppler space for a multi-element linear AESA simulation sampled at 50MHz and the subsequent suppression of these artifacts to less than -100 dB relative to the peak clutter response.

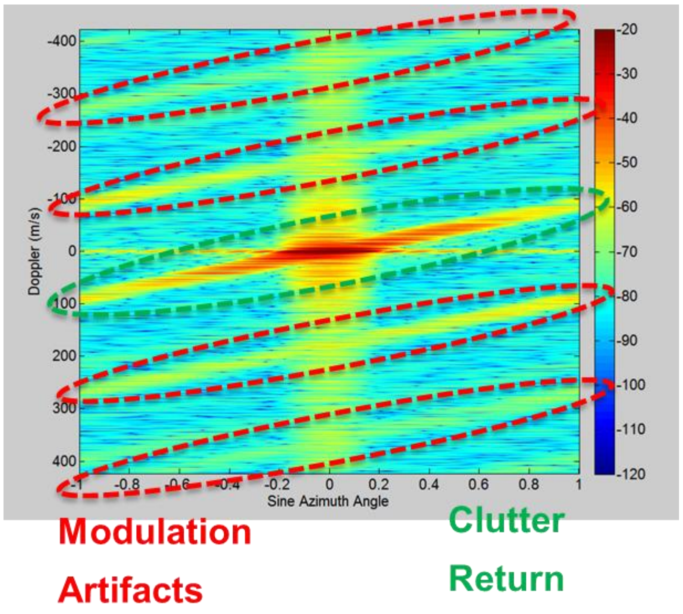

Fig. 2. Original Clutter Model showing model Artifacts

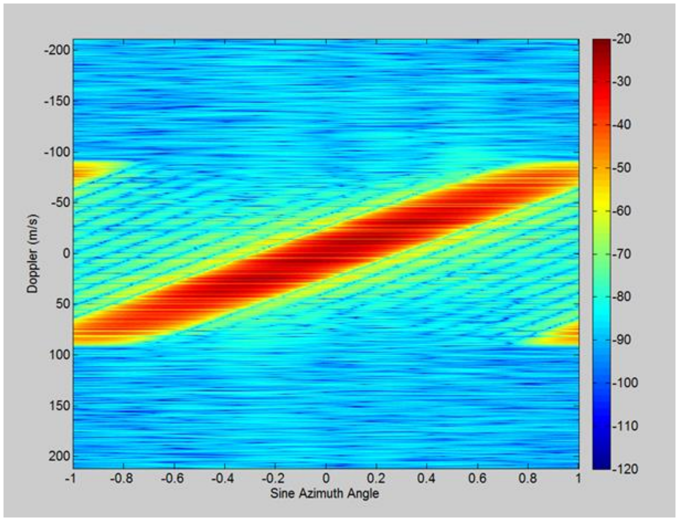

Fig. 3. Ultra-high fidelity model

A baseline simulation was developed using MATLAB® and run on a 16 core Xeon workstation[6] using the MATLAB® Parallel Computing Toolbox. A modest-sized simulation for a 24-element antenna system required 18 days to simulate. With over 100 different simulation iterations to run to verify the system design envelope, the MATLAB-based simulator would not provide results quickly enough to complete the necessary design verification activities. Building on prior work performed through implementing a real-time, synthetic aperture RADAR (SAR) processor in GPGPUs, SRC ported the MATLAB simulation model to NVIDIA CUDA[3]. Initial benchmarking of the GPU simulation indicated a 2.75x speedup from the MATLAB® model.

Even with the 2.75x speedup of the GPU-based simulation relative to MATLAB®, the time per simulation was still too long to provide results in a timely fashion. The next step was to distribute the simulation across multiple GPUs in parallel. The simulation CUDA kernels were integrated into the SRC/AFRL-developed *AirWASP* framework. *AirWASP* is a scalable, distributed, heterogenous framework for high-performance computing applications. *AirWASP* supports multi-process, multi-node, multi-platform, heterogenous distributed computing applications. Additionally, a variety of *AirWASP* applications can be chained together to implement an extended processing chain. The *AirWASP* framework provides a deterministic, routable, data distribution schema that is necessary for applications where a random scatter-gather approach does not work. The *AirWASP* framework uses the ZeroMQ messaging framework. ZeroMQ is an open-source, high-speed, asynchronous messaging library. It supports a variety of common message patterns, such as publish-subscribe, push-pull, and router-dealer over several communication protocols. ZeroMQ is more flexible than the traditionally used Message Passing Interface (MPI).

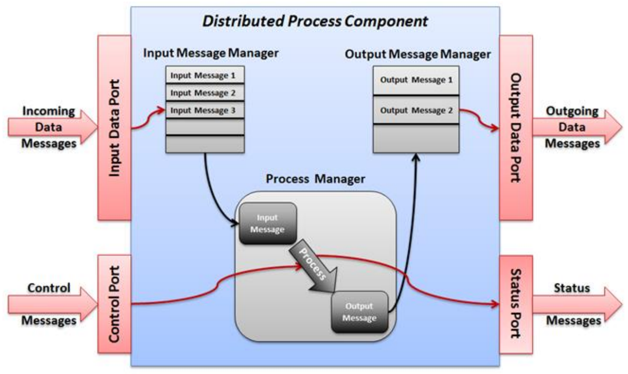

Fig. 4. *AirWASP* distributed processing component

The flexibility and scalability of *AirWASP* has been demonstrated across a range of SWaPs from the AFRL Rome Affiliated Resource Center Offspring supercomputer, to the AgileCONDOR© pod-based High Performance Extreme Computing (HPEC) system[1], [2].

III. METHOD: SIMULATOR ARCHITECTURE AND DESIGN

A. Algorithm

Fundamentally the simulation computes the path length between the emitting element, the scattering element and the receiving element in a three-dimensional Cartesian space. The response is weighted by the antenna gain, which is computed from a full hemispherical, far-field model. The waveform response is computed for each sampling period of the receive element and accumulates those samples responses across the sampling period for all scattering elements in the simulation. Every sampling period the antenna elements positions are updated and the process is repeated.

B. Architecture

Computationally the simulation is a natural fit for the GPU single instruction, multiple-data (SIMD) architecture as each element response is independent from others. Furthermore, the Cartesian geometry maps well to the spatial memory access patterns of the GPU. The primary data dependence is with the environmental scattering responses, and exist only if multiple sampling periods need to be coherently processed. Even then, the simulation can be further parallelized by duplicating the environmental scattering responses across multiple GPUs and processing a subset of sampling periods on each GPU. *Fig. 5* shows an example simulation topology using 3 servers each with a different number of GPUs.

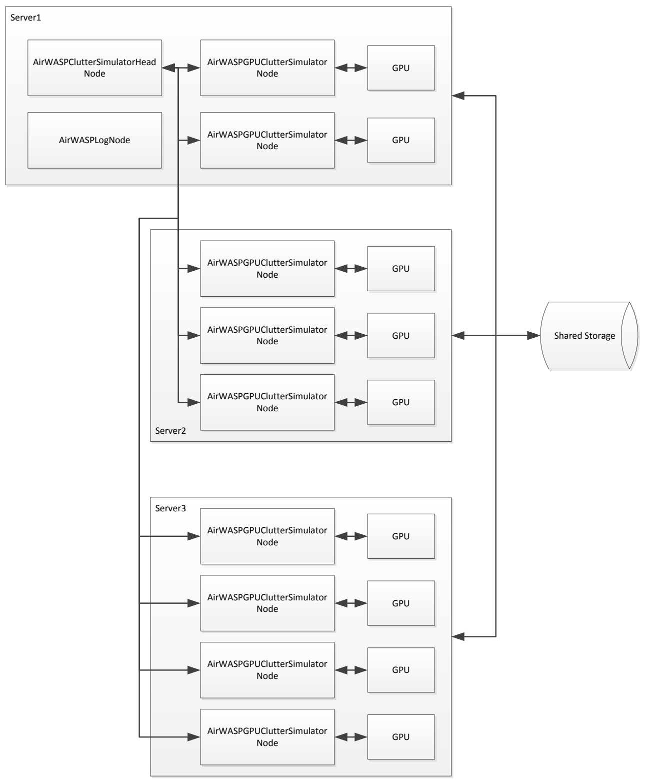
Server1
AirWASPClutterSimulatorHead Node
AirWASPGPUClutterSimulator Node
GPU
AirWASPLogNode
AirWASPGPUClutterSimulator Node
GPU
Server2
AirWASPGPUClutterSimulator Node
GPU
AirWASPGPUClutterSimulator Node
GPU
AirWASPGPUClutterSimulator Node
GPU
Shared Storage
Server3
AirWASPGPUClutterSimulator Node
GPU
AirWASPGPUClutterSimulator Node
GPU
AirWASPGPUClutterSimulator Node
GPU
AirWASPGPUClutterSimulator Node
GPU

Fig. 5. Example Simulator Topology

The simulation geometry assumes the scatters are arranged on a uniformly spaced two-dimensional grid, each grid dimension can have an independent, but uniform spacing, with one dimension representing the along track [forward] motion of the system and the other dimension representing the cross track [right] motion. Elevation about the grid is represented as a scaled, signed offset from a lookup table. This geometry is illustrated in *Fig. 6*. Scatterers on the grid are indexed from the top left corner of the grid, and their Cartesian distance is computed from the grid index using a reference offset method that leverages the fused, multiply-add operations within the GPU.

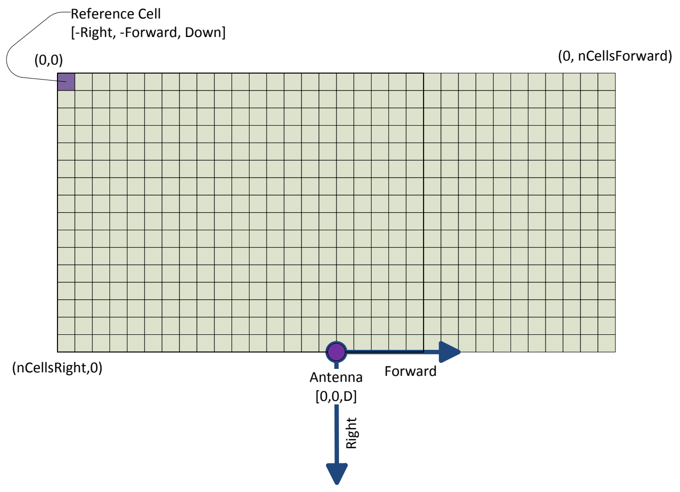

Fig. 6. Simulation Geometry

The CUDA kernels use a blend of integer, single-precision and double-precision floating-point operations. Double-precision floating point operations are used when necessary, primarily for propagation delay calculations. Texture memory is utilized for the waveform response, element beam pattern, and scatterer response memories. The target GPU architecture is the Tesla Kepler series using CUDA 7.5 and the symmetric multiprocessing (SMP) architecture set to 3.5.

C. Design

The simulation consists of two CUDA kernels. It also requires three externally generated inputs; the scatterer reflectivity values, the waveform, and the hemispherical antenna response. These inputs are loaded into texture memories in the GPU.

The first kernel computes the antenna gain value for each scatterer. A simplification is made that assumes that the aspect angle variance from each antenna element to the scatterer is small relative to the antenna beam pattern response in the far field. The primary computation is the direction vector between the antenna phase center and each scatterer. This direction vector is used to perform a lookup into the antenna beam pattern texture memory. This computation and texture lookup is independent for each scatterer. As such, the computations can be parallelized an embarrassingly parallel fashion by simply distributing the scatterers to process across all the available threads. This scheme is illustrated in *Fig. 7*. Thread allocation per block and grid is tuned on a per architecture basis to optimize overall occupancy.

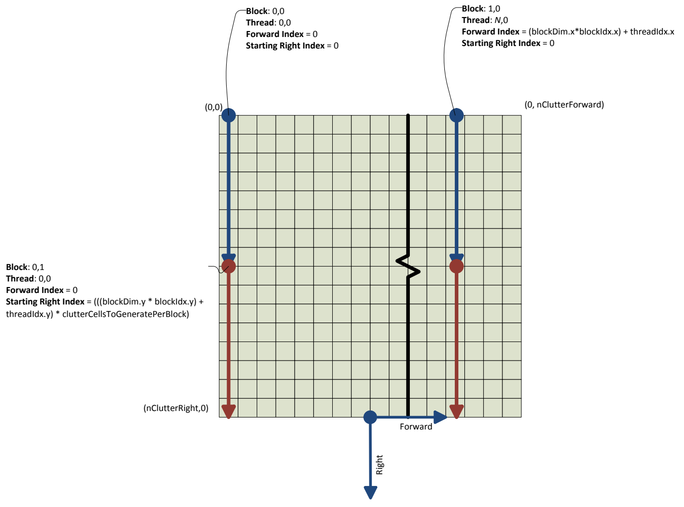
Block: 0,0
Thread: 0,0
Forward Index = 0
Starting Right Index = 0
Block: 1,0
Thread: *N*,0
Forward Index = (blockDim.x*blockIdx.x) + threadIdx.x
Starting Right Index = 0
(0,0)
(0, nClutterForward)
Block: 0,1
Thread: 0,0
Forward Index = 0
Starting Right Index = (((blockDim.y * blockIdx.y) + threadIdx.y) * clutterCellsToGeneratePerBlock)
(nClutterRight,0)
Forward
Right

Fig. 7. Pulse Response Kernel Parallelization

These responses are stored in-situ in GPU memory as a 2D surface. This surface is then converted to a texture memory for the next stage of simulation. These computations are completely independent per scatterer and can be scheduled in the GPU using the maximum number of threads available.

The second kernel computes the contribution of each scatterer to the samples of each antenna element. The computations in this kernel grow multiplicatively in nature. For each scatterer, the path length from each emitting element to a single receive element is computed. Then the contribution of the emitted signal is accumulated into each sample the receive element collects. To help manage memory access, the kernel is parallelized along the receiver element dimension and is aligned to the pitch of the GPU memory access. The parallelization scheme is illustrated in *Fig. 8*.

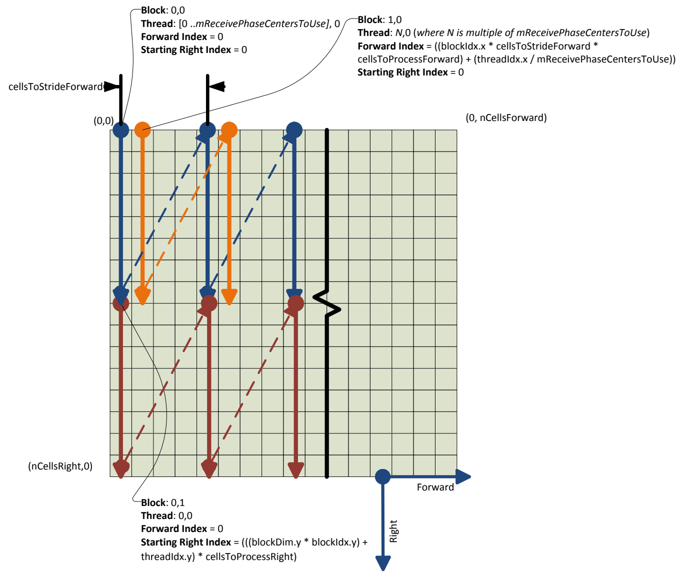
Block: 0,0
Thread: [0 ..*mReceivePhaseCentersToUse*], 0
Forward Index = 0
Starting Right Index = 0
Block: 1,0
Thread: *N*,0 (*where N is multiple of mReceivePhaseCentersToUse*)
Forward Index = ((blockIdx.x * cellsToStrideForward * cellsToProcessForward) + (threadIdx.x / mReceivePhaseCentersToUse))
Starting Right Index = 0
cellsToStrideForward
(0,0)
(0, nCellsForward)
(nCellsRight,0)
Block: 0,1
Thread: 0,0
Forward Index = 0
Starting Right Index = (((blockDim.y * blockIdx.y) + threadIdx.y) * cellsToProcessRight)
Forward
Right

Fig. 8. Range Response Kernel Parallelization

D. Measured Performance

The following figures show the relative performance improvement between the baseline MATLAB® parallelized simulation and the equivalent GPU based simulation using 12 NVIDIA Tesla K20m GPUs on the AFRL Offspring High Performance Computer (HPC) using the *AirWASP* framework. Simulation time has decreased from 16.5 days to less than 1 day.

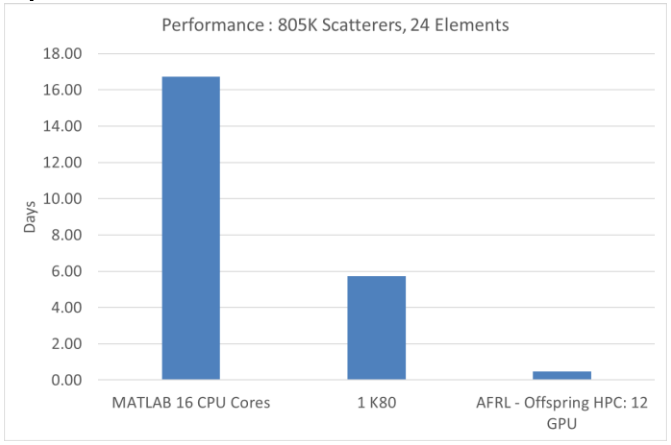

Fig. 9. Simulation performance improvement

IV. RESULTS

The kernels were profiled using the NVIDIA Visual Profiling tool to capture occupancy, operation count and memory throughput. One interesting finding of the profiling is that both kernels are memory bandwidth bound. While the memory accesses within the kernel were optimized as much as possible, the ratio between memory accesses and computation operations is less than optimal.

Metrics were collected to profile the timing as a function of elements, samples per element and scatterers simulated. *Fig. 10* shows the increase in simulation time per antenna element as the sample count increases. The time growth is roughly log linear.

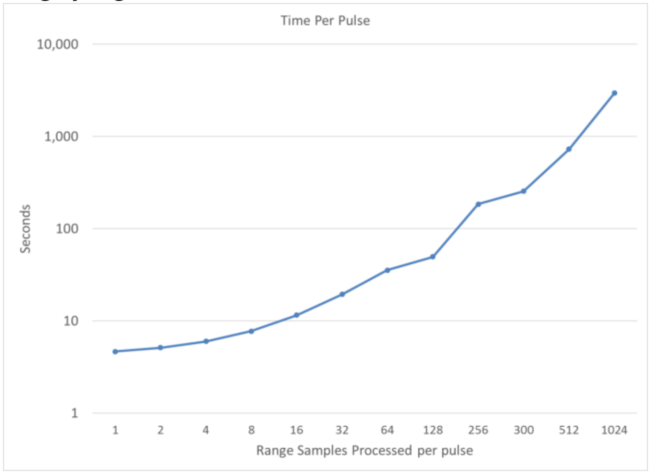

Fig. 10. Simulation Time as a Function of Samples collected per element

Fig. 11 shows the increase in simulation time as a function of scatterers modeled for a constant sample count. The time growth is roughly linear.

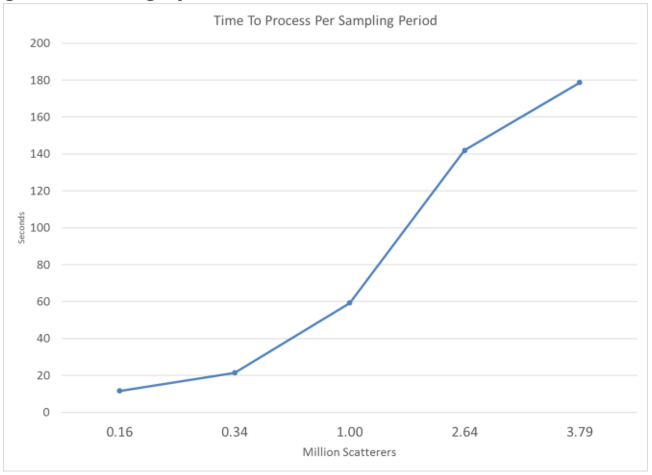

Fig. 11. Simulation Time as a function of scatterer count

An interesting finding during early testing of the simulation was that depending on the simulation geometry, the number of actual scatterers per sampling period that were necessary to process dropped as the simulation geometry increased. This is due to the overall size of the rectangular grid used to represent the simulation scene and the actual number of scatterers that are illuminated by both the emitting and receiving element for each sampling period. This is illustrated in *Fig. 12*.

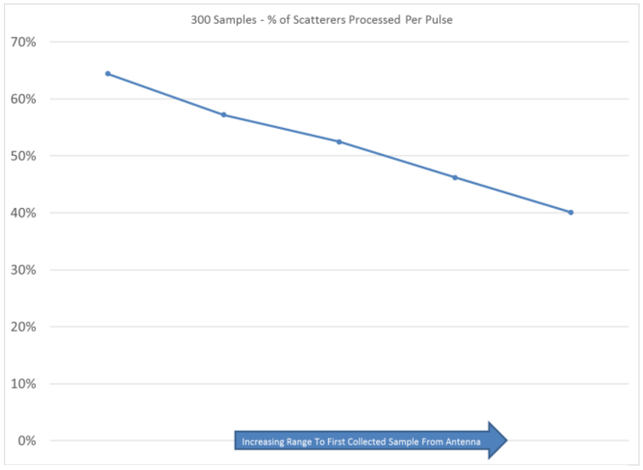

Fig. 12. Percent Scatterers processed

As a result of these findings, the simulation was further enhanced to operate on smaller rectangular blocks within the overall scene that capture only the scatterers that are illuminated for a given sampling period. This is illustrated in *Fig. 13* for a scenario collecting 300 samples at several kilometers distance from an antenna with roughly 12 degrees of beamwidth.

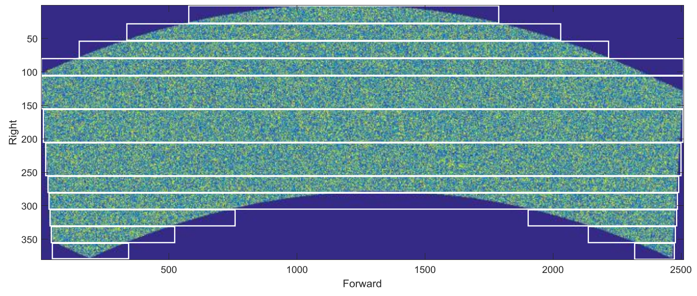

Fig. 13 Illuminated scatterer selection illustration

V. Conclusions

The computations necessary to model the RF propagation of a modern RF system are truly exa-scale. Without the use of GPGPUs and distributed heterogenous computing, these simulations would be impossible to complete on a timeline that would provide actionable information necessary to design the next generation of RF system.

Utilizing the AFRL Offspring HPC to run the GPU-based clutter model has reduced simulation time by a factor of 16, which will allow design verification activities to be completed in a timely fashion. This joint partnership between government and industry is enabling future RF systems to be designed in a timely and cost effective fashion.

VI. Future Work

Future work to the simulator could include capabilities for simulating systems where the emitter and receiver are not closely co-located. The current simulation assumes the element beam pattern response for the elements in the system is the same for each scatterer. This is not true if the element locations have significantly different aspect angles to a given scatterer.

Acknowledgment

The authors would like to thank Darrek Isereau, Christopher Capraro, Lun Chen and Christopher Cicotta for their contributions. . Received and approved for public release by AFRL on 10 August 2017, case number 88ABW-2017-0863. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors, and do not necessarily reflect the views of AFRL or its contractors.

References

[1] C. L. Usmail, M. O. Little and R. E. Zuber, "Evolution of embedded processing for wide area surveillance," in IEEE Aerospace and Electronic Systems Magazine, vol. 29, no. 1, pp. 6-13, Jan. 2014.

[2] R. E. Zuber, C. Capraro, M. Barnell, M. Little, "An Accelerated and Agile Exploitation Framework Leveraging GPUs: Airborne Wide Area Synthetic Aperture Radar Processing (Air WASP)," 2013 GPU Technology Conference, San Jose, CA, 2013.

[3] (2017, May 24). *NVIDIA Cuda reference* [Online]. Available: https://docs.nvidia.com/cuda/index.html

[4] (2017, May 24). *NVIDIA Cuda reference* [Online]. Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#introduction

[5] (2017, July 20). *AN/SPY-1 RADAR* [Online]. Available: https://en.wikipedia.org/wiki/AN/SPY-1

[6] Dual Intel Xeon Romley E5-2690 CPU 2.9GHz with 128GB SDRAM running Windows Server 2012