High-fidelity Real-time Antiship Cruise Missile Modeling on the GPU

Christopher SCANNELL[a,1], Jonathan DECKER[a] , Joseph COLLINS[a] and William SMITH[b]
[a] *Naval Research Laboratory, Washington, DC*
[b] *ITT Defense & Information Solutions, Mclean, VA*

Abstract. The United States Navy is actively researching techniques for creating high-fidelity, real-time simulations of antiship cruise missiles (ASCM) in order to develop improved defensive countermeasures for Navy ships. One active area of investigation is the combined use of OpenMP and MPI to reach real-time constraints on stand-alone cluster computers with high-speed interconnect fabrics. The separate compute nodes of the supercomputer calculate the successive responses of a single cruise missile to successive reflections of the RF transmitter radar returns from the target ship in a pipeline fashion using MPI. Numerically intensive portions of the calculation of the missile-ship system behavior for an individual RF pulse can be calculated in parallel simultaneously on the individual nodes of the supercomputer using OpenMP. The speed at which these portions can be calculated directly determines the length of the pipeline and thus the total number of computing nodes required. This approach incurs some approximations into the simulation that are proportional to the length of the pipeline because there is a feedback from the ship-radar response back to the missile guidance. While this use of OpenMP has proven effective, it is limited by the number of cores available at each node. This code, however, presents opportunities for parallelism well beyond the available computational resources at each node. Additionally, the ratio of computation to data transfer for this portion of the simulation is very high. These two factors have led us to investigate executing the most compute-intensive portion, the calculation of the RF responses of the individual ship scatterers, on Graphics Processing Units (GPUs).

Keywords. GPU, OpenCL, antiship, cruise, missile

Introduction

The Naval Research Laboratory (NRL), located in Washington, DC, is the US Navy corporate R&D center, executing basic and applied programs over a broad range of science, technology and mission applications. This paper focuses on work being done in an NRL project to investigate and exploit Graphics Processing Unit (GPU) technology for the benefit of compute intensive Naval applications. As part of this work, a large physics-based computer simulation model of ship missile defense systems, known as CRUISE_Missiles (Ref. 1), was chosen as a GPU prototyping application (see Figure 1). CRUISE_Missiles is apropos because the US Navy has been actively researching techniques for achieving real-time simulation to help develop

[1] Corresponding Author. Tel.: +202-767-1127; fax: +202-767-1122.
Email: Christopher.Scannell@nrl.navy.mil

improved defensive countermeasures for Navy ships. One active area of investigation for simulation performance enhancement has been the use of computer clusters communicating via Message Passing Interface (MPI) over high-speed interconnect fabrics such as InfiniBand.

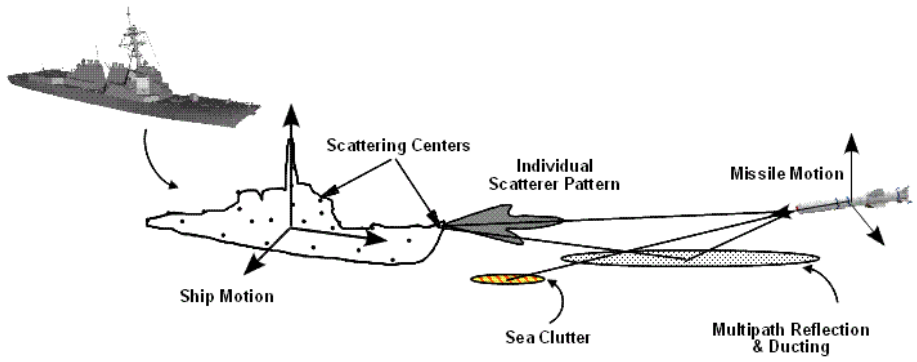

Figure 1. System elements of CRUISE_Missiles simulation.

Separate computer central processing units (CPUs) model the behavior of the cruise missile at discrete instants in time at the frequency of the pulse repetition of the missile's active radar. Each CPU calculates the behavior of the missile for its particular pulse repetition interval (PRI) and passes the resulting state of the missile to the CPU simulating the missile for the subsequent PRI. After handing off this state information, the CPU calculates the response of the target to the radar frequency (RF) energy that was just transmitted. The response calculation is the most compute intensive portion of the simulation and currently consumes twenty to thirty times more wall-clock time to compute than the missile update on modern multi-core CPUs. Fortunately, the time required to model changes in the state of the missile from one PRI to the next can be kept to less than the pulse repetition interval, so the simulation is able to run in real-time if there is a sufficient number of CPUs in the cluster and if the required fidelity of the simulation can tolerate the effective delay in the computed state of the RF return signal. The target response is calculated independently on each CPU (using the latest missile state information available to the CPU) but this response is not communicated with the other CPUs. Instead, this computed target response is used twenty to thirty PRIs later when the CPU is required to compute its next missile state update. This effective lag of twenty to thirty PRI in the perceived state of the target by the missile is acceptable because the rate of change of the target is much less than that of the missile.

The parallelism inherent in this target response calculation is currently being exploited using OpenMP. While OpenMP has proven effective, it is limited by the number of cores available at each node relative to the number of independent ship scatterer elements. That, and the fact that the ratio of computation to data transfer load for this portion of the simulation is very high, has led us to focus our current effort on studying how General Purpose Computation on Graphics Processing Units (GPGPU) techniques can be applied to speed up the target signature calculation to reduce the size of the MPI ring of processors in our simulation.

The Physical System

The real-world system of interest comprises several key interacting elements: the missile, one or more targets (i.e., ships) and a marine environment (ocean and atmosphere). The missile includes several subsystems: radar emitter, receiver and video processor subsystems, and propulsion and guidance subsystems. A target ship, as seen by the missile radar subsystem, is composed of multiple (thousands to millions) reflecting RF scatterer centers, which can be categorized according to physical structure (e.g. point, plane, cylinder, dihedral). The ocean constitutes a reflecting and scattering surface to the radar RF waves, and the atmosphere a diffracting medium. Figure 1 illustrates the system elements.

In operation, the system constitutes a closed loop process. The missile emits consecutive radar pulses at times t_i. Each pulse's energy propagates to the ship by direct and by indirect ocean-surface-reflected paths, then is reflected from the ship scatterer centers back to the missile, again by direct and indirect paths, where the received signals are processed by the video and guidance subsystems to steer the missile toward the target (see Figure 2). The subset of scatterers on the ship that the missile 'sees' at a given t_i is a function of the relative position and orientation of the ship with respect to the missile position.

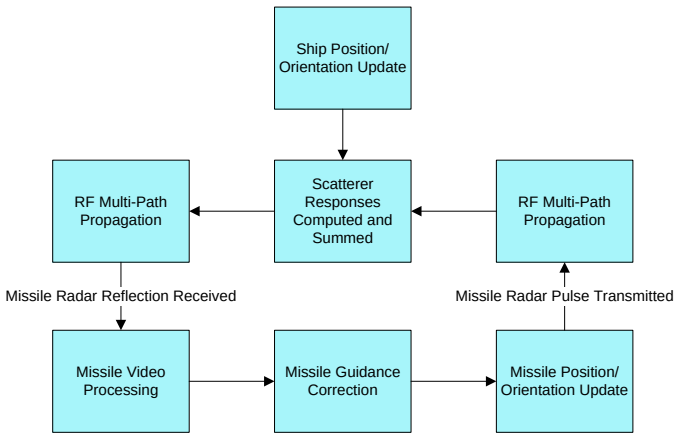

Figure 2. Ship-missile behavioral data flow.

The CRUISE_Missiles Simulation

CRUISE_Missiles represents a body of models and computer codes that allows for the detailed physics-based modeling of multiple types of missiles, multiple types of targets, and their missile-target interactions. The requirements of the simulation specify a high degree of realism by using physics-based modeling for the purpose of achieving valid predictions of missile-ship interactions in the marine environment. The NRL CRUISE_Missiles missile-ship simulation model suite provides the base library of valid models for this purpose.

The CRUISE_Missiles simulation system was developed under different names over a period of several decades for representing target, threat, and countermeasure platforms in a marine environment. The system's modularity allows one to simulate the platforms by executing simulation object state updates on very short timescales. A simulation run entails many iterations of these short time-scale behaviors, where validity is maintained by making sure that the updates are executed in a causal sequence with correct dependencies observed. The original execution model in CRUISE_Missiles was a serial execution model, that is, the tasks of updating the short-time scale state updates are executed on a single serial processor.

CRUISE_Missiles is an event-driven simulation advanced on a pulse-by-pulse basis, where a pulse is the active radar emission from the missile. The PRI is typically in the sub-millisecond range. For each pulse, the target "image" or "video" scene, as perceived by the missile, is modeled by computing the responses of the target ship's visible scatterers to the pulse via direct and indirect propagation paths. Additionally, the missile response to the "image" or "video" signal is also computed. Signal representation and computations are in the discrete time-sampled domain. Three stages of processing are required for computing the video signal: (1) convolution of the transmitted radar pulse function, $Tr(t_i)$, with the impulse response functions of each individual independent scatterer, $S_k(t_i)$, and its associated incident and reflected propagation path, $Pi_k(t_i)$ and $Pr_k(t_i)$, (2) application of the range gate windowing function $Rg(t_i)$ to select a common time subset (e.g., 100 samples) from the returned signal of every scatterer, and (3) coherent summation, over all the scatterers, of the range-gated samples to produce the "Composite video" signal $V(t_i)$.

$$V(t_i) = \sum_{Scatterers\ 1\ldots k} [Tr(t_i) \otimes Pi_k(t_i) \otimes S_k(t_i) \otimes Pr_k(t_i) \otimes Rg(t_i)]$$

In the serial implementation of CRUISE_Missiles, the computation sequence repeats in a serial loop (figure 2 illustrates).

CPU Cluster-Based Parallelization of the Model

Prior to OpenMP parallelization, CRUISE_Missiles modeled missile/target interactions at a rate of a few percent of real-time. NRL undertook increasing the performance of CRUISE_Missiles for the purposes of operating in a real-time environment where CRUISE_Missiles interacts with other models in a High-Level Architecture (HLA) federation (see Ref. 2). Taking these requirements into consideration, NRL reviewed how one might address the fundamental problem of speeding up the simulation. This multi-year effort consisted of various activities: discussions and design sessions between parallel programming and model experts, research of current literature describing hardware features relevant to speeding up CRUISE_Missiles, analysis and performance profiling of simulation code, and implementation and testing of code modifications to prove the feasibility of our proposed redesign.

The ship scatterer and multipath RF pulse response modeling - that is, the target response computation - constitutes the major portion of the computational loop load and, therefore, the primary objective for speedup. There are many scatterers visible and each scatterer may be composed of many components (e.g. dihedral, trihedral, N-

hedral). Since the calculation of the contribution to the target signal of these reflections (for a particular position and orientation of the missile) can be performed entirely independently of all other such calculations, there exists the potential to independently evaluate component elements of the target response in parallel, prior to combining them for the missile video process. Because the ship signature changes slowly with respect to the radar pulse rate and the missile control dynamics, processes related to this were identified which could be delayed to achieve real-time performance without undue degradation in simulation fidelity. Other loop processes would have to be completed without delay every pulse. To illustrate, figure 3 shows a basic decomposition of the computation of the missile-target system showing the time sequence of the missile update and target response functions.

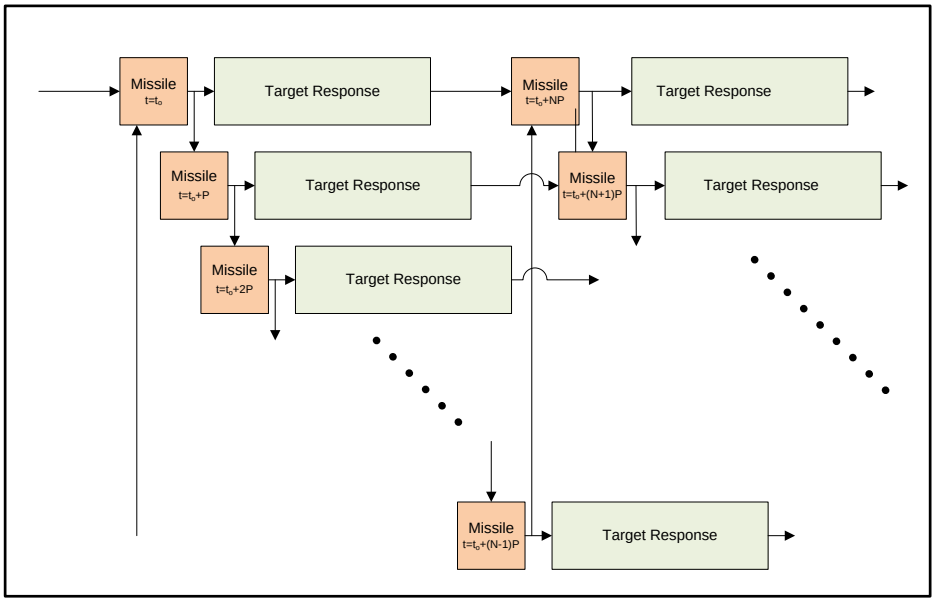

Figure 3. Data flow diagram/Parallel decomposition diagram.

The missile update function computation must complete on each pulse, without delay, in time to trigger the next missile update. However, the target response, due to the slow change of the ship signature, can be delayed without degradation to simulation fidelity. This means that a target response corresponding to time t_i, instead of being passed to the missile update function at time t_{i+1}, can be passed to the missile update corresponding to time $t = t_{i+N}$, N pulse times later. So, instead of speeding up the computation of each target response instance, N overlapped and delayed instances, corresponding to successive pulse times, are computed independently and concurrently (see Figure 3). N of these overlapped instances form a ring, with effective process speedup of N and loop delay of N * PRI. Each of the MPI-connected nodes in the ring are assigned to separate multi-core CPUs of a cluster computer with typically up to thirty nodes required for a modest simulation scenario to maintain real-time performance. Due to the independent nature of the calculation, considerable speedup has been achieved (~80% of maximum linear speed up equal to the number of cores of each node's CPU) using OpenMP.

GPU-Based Parallelization of the Model

The aspect of the target response calculation that permits significant speedup using OpenMP (i.e. the independent and compute-intensive nature of the computation of the responses of the individual scatterers that compose the target) can be even more aggressively exploited by the use of GPU technology. This is due largely to four essential facts: 1) the response of thousands of independent scatterers need to be calculated for each pulse, 2) very little pulse-to-pulse simulation state information influences the calculation so little data needs to be transferred from the CPU to the GPU per kernel invocation, 3) the output of the target response calculation can be compactly represented as a short timeseries signal, so that very little data needs to be transferred from the GPU to the CPU per kernel invocation and 4) significant time-invariant data (e.g. attenuation factors for different scatterer materials, vertex coordinates of plates shared by multiple scatterers) is shared by scatterers which allows for GPU speedup techniques to be applied that involve coalesced loads of shared data from global to local memory. Our initial CUDA and OpenCL implementations of the target response calculation confirmed the GPGPU applicability of this computation (see Results section). These implementations were tested and validated for accuracy, with differences in the signature that were limited to the least significant bits. This is of particular importance for this application area due to the high-fidelity requirements of the model (confirmed using massive Monte Carlo test validation runs).

To-date our implementation is limited to one type of scatterer. We began with the N-hedral scatterer because it constituted the majority of the computational load for the target signature generation both in terms of numbers of scatterers and in scatterer complexity. An N-hedral is composed of N plates that mutually reflect the RF energy among themselves and back towards the radar through a prescribed and ordered set of bounces. The radar cross section contribution of each N-hedral is very sensitive to the exact relative position of the missile to the ship and is not calculated as a closed algebraic computation but rather using ray-tracing techniques. For a given azimuthal/elevation sector of the airspace relative to the target, pre-computed lists of N-hedrals are identified as potentially contributing to the target signal. The N-hedrals that are visible for given orientation of the ship relative to the incoming missile are composed from a base set of plates. Because the size of this base set is significantly smaller than the size of the set of visible N-hedrals, we can make efficient use of the GPU local memory to simultaneously store the scatterer state data that underlies a given set of N-hedrals. To take best advantage of this feature of N-hedrals we must sort the N-hedrals according to their constituent plates to improve locality of reference in the GPU local memory. Another shared attribute of scatterers that can be leveraged through explicit GPU memory management is the material property characteristics that are shared across all instances of scatterers.

Results and Future Work

An important objective of this in-house NRL research effort is to provide guidance to Navy model experts in the use of effective GPGPU programming practices, including the tradeoffs of using CUDA versus OpenCL. A direct CUDA implementation will only run on NVIDIA graphics-hardware which is CUDA-complaint. OpenCL is very similar in syntax, but can compile to a multitude of platforms, including multi-core

CPUs and AMD graphics hardware. However, CUDA has hooks that allow the programmer to work closer to the specific hardware, which may incur additional speed-ups. Additionally, CUDA development had a head start on OpenCL development, and as a result there are a number of development libraries available which implement various standard parallel algorithms.

The entire legacy implementation of the N-hedral scatterer computation for the target response was ported to the GPU using both CUDA and OpenCL and the results validated against the original CPU C/C++ implementation. There were two significant elements of this effort. The first one involved the restructuring of the hierarchy of the control loops of the C/C++ code so that outermost control loop was over scatterers. The legacy code was organized as a series of numerous functions, each which was carried out in turn over the entire set of scatterers. The second element was the restructuring of the legacy data structures as one-dimensional buffers and the management of sending and receiving these buffers between the CPU and GPU.

A naive port of the code would overwhelm the respective high-level language compilers, since there is only so much hardware (memory and registers) available to each individual thread on the GPU. CUDA proved to be easier to work with than OpenCL because of its relative maturity and the convenience of being able to invoke print operations directly from the kernel for debugging purposes. We accomplished this by utilizing the cuPrintf capability demonstrated in the CUDA SDK. Figure 4 shows the comparative performance of the GPU implementations relative to a single-core CPU.

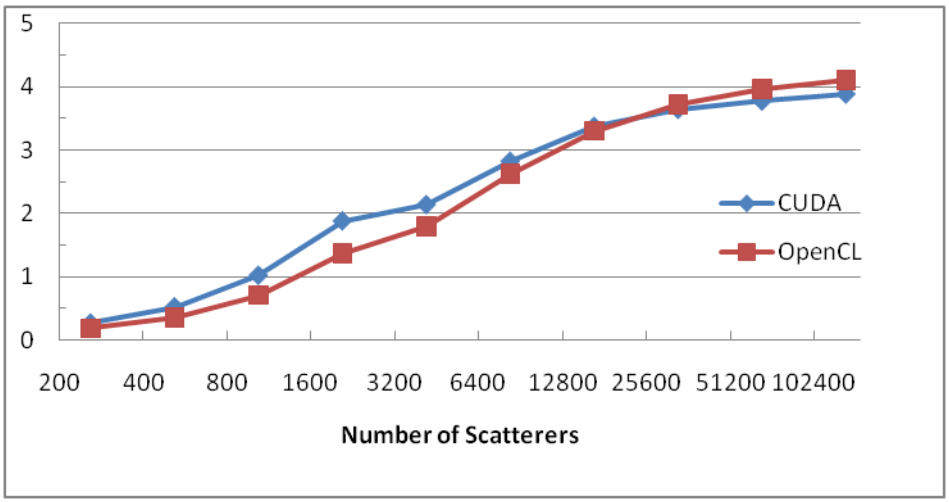

Figure 4. Speedup achieved running scatterer calculations for CUDA and OpenCL.

This speedup displayed in figure 4 constituted the initial results of our study absent any optimizations regarding explicit management of the memory hierarchy, scatterer execution ordering, reduction of integer and branching operations, or use of constant memory. Judging by the results of other GPGPU applications, we expect to achieve at least an order of magnitude increase in speedup from these types of optimizations. Additionally, there are a few other types of scatterers that remain to be addressed for our current targets of interest. We have yet to implement the efficient summation of

the results of the individual scatterer responses, but techniques for performing such a reduction operation are well understood. Finally, the video compositing portion of the legacy CRUISE_Missiles software involving windowing the timeseries to account the missile range gate needs to ported to the GPU.

The work to date has been performed using an NVIDIA GeForce GTX 285 on a 32-bit X86 platform running LINUX. We are currently collaborating with Dell Inc. to investigate how we might we combine C2070 Tesla hardware (Tesla is a line of hardware created by NVIDIA which is specifically designed to accelerate general-purpose applications using CUDA) with our conventional MPI-connected CPU cluster computing platform. We are interested in how the resulting hybrid architecture would impact performance and fidelity as compared to the existing system.

References

[1] Goldberg, A. J. and Futato, R. J. "CRUISE_Missiles Electronic Warfare Simulation." NRL Review (1993): 123-126. Print.

[2] Dahmann, J. S., Fujimoto, R., Weatherly, R. M. "The Department of Defense High Level Architecture. Winter Simulation Conference (1997): 142-149. Print.