Georgia Tech Research Institute
Applied Systems Laboratory

Real-Time GPU Computation of Ballistic Thermal Signatures

Dr. Glenn A. Parker
Principal Research Engineer

Background and Problem

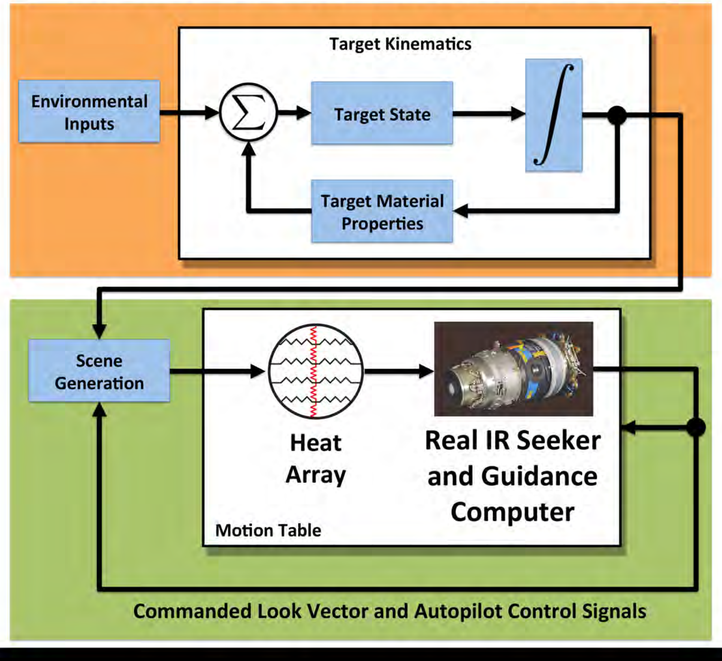
- Extensive hardware tests of single-use systems like missiles can be prohibitively expensive, and pure simulations are unreliable
- If some subsystems are reusable, a hybrid approach called Hardware-in-the-Loop (HWIL) is adopted where hardware is "looped in" with simulated subsystems
- Hardware requires real-time signals, which may change hundreds or thousands of times per second
- Targets are usually missiles, reentry vehicles, or countermeasures
- Since targets may be hypersonic and tumbling, high sampling frequencies are indicated to avoid simulation aliasing
- Depending on the target's geometric complexity, computing the entire simulation may take up to **3 days** for a 20 minute scenario
- Due to such delays, target data is computed offline and stored in a database for lookup during simulation
- **GOAL: We desired at least "near real-time" computation to facilitate many more test runs in a given period**

HWIL Missile Simulation: How It Works

- Missile HWIL using infrared (IR) seekers requires thermal emissivity for the target, which is a function of temperature
- Temperature is a function of time, position on the target, aerothermal heating, and target material properties
- At each simulation time step, solution of the heat equation is necessary to compute target emissivity for input to the seeker
- Surface properties typically change as material ablates (burns off)

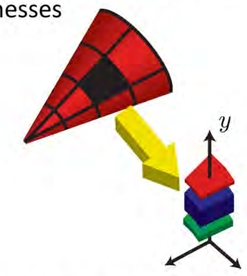
Traditionally very slow; performed offline and stored

Inherently real-time matrix math; no speedup necessary



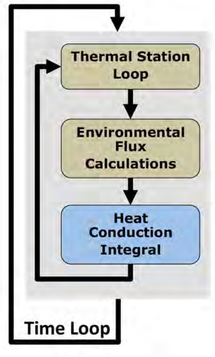
Our CUDA Solution

- In the existing algorithm, a 3D mesh discretizes the target object into a maximum of 500 quadrilaterals
- Each mesh quadrilateral is called a "thermal station"
- Each thermal station is composed of up to 40 layers of differing materials and thicknesses

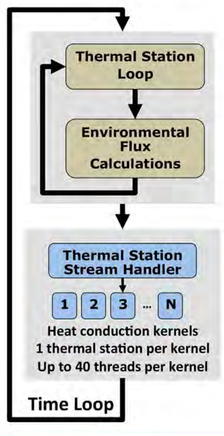


- Our efforts focused on a single-threaded loop in which the existing FORTRAN algorithm solved the one-dimensional heat equation for every thermal station at each time step
- We created a C++ wrapper using CUDA streams to simultaneously solve all thermal stations (up to 500)
- Each stream launches a kernel with up to 40 threads (one for each material stackup layer)
- Each thread solves the heat equation across one material layer

Single-threaded FORTRAN computation



Our CUDA stream approach handles up to 500 concurrent kernels



One-dimensional heat equation:

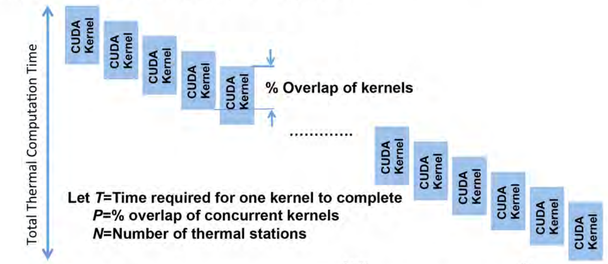
$$\underbrace{\rho C_p \frac{\partial T}{\partial t}}_{\text{energy exchange}} = \underbrace{\frac{\partial}{\partial y}\left(k\frac{\partial T}{\partial y}\right)}_{\text{conduction}} + \underbrace{\dot{m}\frac{\partial h_g}{\partial y}}_{\text{gaseous exchange}} + \underbrace{\Delta H_{dp}\frac{\partial \rho_{dp}}{\partial t}}_{\text{decomposition exchange}} + \underbrace{\dot{s}\rho c_p \frac{\partial T}{\partial y}}_{\text{moving coord system exchange}}$$

Boundary condition on the target surface:

$$\underbrace{-k(\nabla T)_{\text{out}}}_{\text{inward flux}} = \underbrace{Q}_{\text{external sources}} - \underbrace{\varepsilon\sigma T^4}_{\text{exiting radiation}}$$

Results

- We implemented the CUDA Stream algorithm and tested it using a single TESLA C2070
- There is computational overhead with setting up each kernel to run in a stream, and the hardware is not capable of running all 500 streams concurrently
- Based on a preliminary analysis using average kernel times and the number of streaming multiprocessors in the C2070, we expected an "effective concurrency" of at least 80%, resulting in a speedup factor of between 5 and 10 with the maximum number of streams as shown below:



$$\text{Thermal Computation Speedup} = \frac{NT}{T + (N-1)(1-P)T} = \frac{N}{1 + (N-1)(1-P)}$$

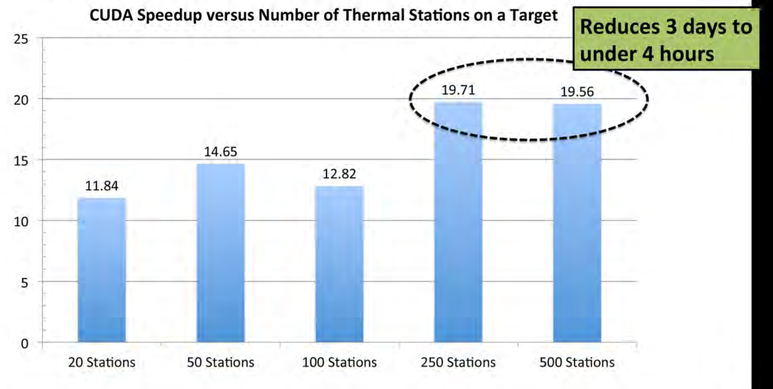
- 500 Thermal Stations
- 80% overlap

$$\text{Speedup} = \frac{500}{1 + 499(1 - 0.8)} \approx 4.96$$

- 500 Thermal Stations
- 90% overlap

$$\text{Speedup} = \frac{500}{1 + 499(1 - 0.9)} \approx 9.82$$

- Actual results were far more encouraging:



What's Next?

- We are fighting stream concurrency limitations at 250 stations, so we want to implement a multiple-GPU algorithm for HWIL
- Simple targets don't need a lot of thermal stations, so we want more performance for those cases
- Find better ways to utilize CUDA
  - "Ghost threads" to fill warps
  - Multiple stations per kernel
  - Coalesce more global fetches
  - Clever uses of shared memory

HWIL testing in progress