2017 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
SYSTEMS ENGINEERING (SE) TECHNICAL SESSION
AUGUST 8-10, 2017 - NOVI, MICHIGAN

ENDGAME SIMULATION USING AN ALL-INCLUSIVE, ULTRA-FAST VULNERABILITY AND LETHALITY CODE

Alexander Bernardo
SURVICE Engineering Company
Troy, MI

Patrick Buckley
Matt Perini
SURVICE Engineering Company
PMC Operation
Socorro, NM

ABSTRACT

An endgame, vulnerability/lethality code, TurboPK was developed to take advantage of parallel processing of multi-core, modern-day desktop and laptop computers. TurboPK is used to simulate and analyze weapon-related kinetic energy and blast effects of military vehicles. It implements Department of Defense (DoD)-approved algorithms and is compatible with the DoD design trade-off process. Its speed advantage is commensurate with the increase in number of cores used. A quad-core processor results in run times that are four times faster than using a single core. The heart of endgame analysis calculates geometric intersections of projectiles or fragments with vehicle components using ray-tracing algorithms. For example, literally thousands of rays are used to accurately model the fragment ejecta from a warhead in a burst point analysis. Algorithms originally written for a single processor have been rewritten to exploit an open-source, parallel process ray tracer called Embree, provided by Intel Corporation.

INTRODUCTION

In support of ground vehicle vulnerability assessment efforts, a highly optimized code for modern desktop and laptop central processing units (CPUs) has been developed and is currently being implemented. The code, named TurboPK, was developed by the SURVICE Engineering Company. It exploits multicore, superscalar, hyperthreaded, and vectorized processes and is a traditional endgame code that runs on both Microsoft Windows and Linux platforms.

The code can be used to simulate and analyze weapon-related kinetic energy and blast effects, including armor-piercing projectiles, fragments, exploding munitions, and air blast. Simultaneously, it is integrated within the traditional design trade-off process, ultimately resulting in an optimized ground vehicle, providing the user with a rapid ability to determine a system's probability of kill (Pk), and ensuring traceability to DoD and industry standard methodologies and algorithms.

The analysis implementation takes advantage of numerous familiar input formats, including Geometric Model, COVART4 Component Damage Functions, COVART4 Fault Tree, Projectile File, and Warhead File formats. In implementation, the geometric model is directly derived from standard computer-assisted design (CAD) geometry using the STereoLithography

(STL) format. Using the failure mode and effects analysis (FMEA)/failure modes, effects, and criticality analysis (FMECA) approach and resulting fault trees, multiply vulnerable components are addressed. Component damage functions are input via empirical probability of kill given a hit (Pk/h) curves.

In a TurboPK analysis, all vulnerable components are included, including crew, engine/propulsion system, suspension, steering, hydraulic, pneumatic, cooling, and lubrication systems. The analysis process is greatly streamlined, as TurboPK has a built-in library of materials for the penetration equations. SURVICE also has a library of empirical Pk/h curves for vulnerable components.

Within TurboPK, widely accepted algorithms are implemented, such as those that originate from the Joint Technical Coordinating Group for Munitions Effectiveness (JTCG/ME) Penetration Equations Handbook and that use the COVART4 Component Damage Functions and the COVART4 Fault Trees. In addition, cases have been run using an integrated third-order polynomial equation for air blast.

EXPLOITING PARALLEL PROCESSING

Today, virtually every desktop, laptop, and notebook computer is built for parallel processing. But most legacy endgame software were not originally written for parallel processing, so the full power of these multi-core CPUs went unused.

Fortunately for weapons analysts, endgame codes are highly amenable to parallel processing and can take full advantage of multi-core CPUs to increase modeling and simulation (M&S) speed. New capabilities that were not practically feasible before, such as near-real-time design optimization, can now be developed.

Through investigation and implementation of various parallelization schemes and employment of various software development tools, SURVICE was able to demonstrate that parallelizing endgame codes is practical in terms of providing impressive reductions in simulation run times and that these improvements scale linearly over a small number of cores.

In the course of vulnerability/lethality analysis, an analyst might be interested in the damage effects of fragments from a warhead detonation on a target. In this case, a burst-point analysis can be conducted in TurboPK. Multiple burst points can be run quickly where a Pk contour can be generated over a geometric plane relative to the target. In a burst-point scenario, one obvious way to speed up the analysis is to subdivide it into subsets that then run in parallel on multiple cores.

The scheme employed in TurboPK is illustrated in Figure 1. To implement this scheme, TurboPK creates a separate computational thread for each core, asks the operating system to launch the threads, waits for all the threads to complete their work, and then merges the results from all the threads. Each thread is a complete point-burst program. Point-burst simulations model a fragment warhead burst in Monte Carlo fashion as a set of fragment rays whose directions and speeds are randomized according to a prescribed distribution function.

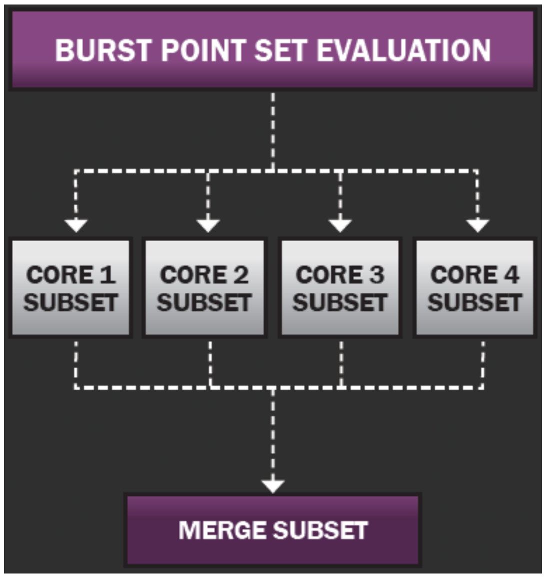


Figure 1: Typical multi-core parallelization scheme.

An example point burst is depicted in Figure 2 as a set of fragment shotlines emanating from a point located a few meters above the target model of interest. In a point-burst code, each shotline is first ray traced against the target geometry model in question to determine which geometry objects, if any, it intersects. The ray-tracing step is computationally intensive and often consumes 80% of the runtime in a point-burst code due to the large number of potential ray-object intersection tests involved.

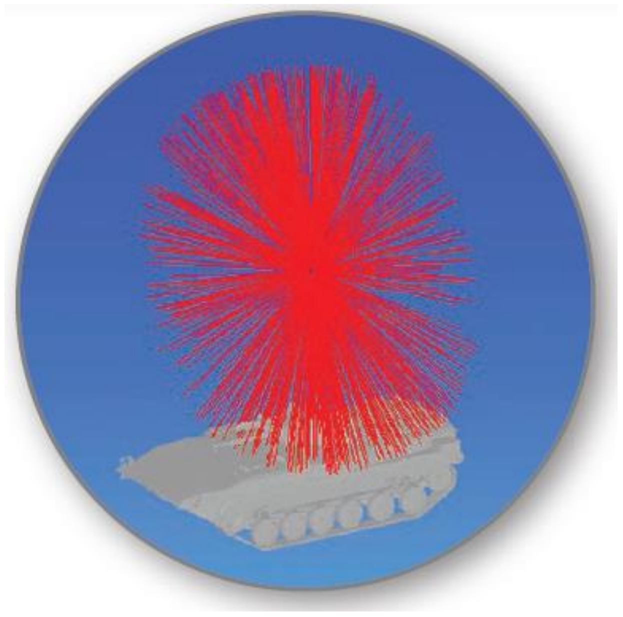


Figure 2: Example of a warhead point-burst simulation.

Consider, for example, a warhead that ejects 2,000 fragments and a geometric model that has 100,000 triangles. In theory, that results in 200,000,000 ray-triangle intersection tests to be performed per point-burst calculation. Many such calculations are performed in a typical analysis session, so it is easy to see why ray tracing dominates the run time.

Fortunately for endgame programmers, ray tracing has been the subject of a great deal of research, and there are numerous high-quality, open-source ray tracers that greatly reduce ray tracing times for endgame codes.

The current default ray tracer is an open-source ray tracer called Embree [1], provided by Intel Corporation. Embree fits the scheme illustrated in Figure 1 because it is “thread safe.” That is to say, the code manipulates only shared data structures in a manner that guarantees safe execution by multiple threads at the same time. Embree also employs a low-level type of parallelism in the form of single instruction multiple data (SIMD) instructions. Among other things, SIMD enables Embree to test one ray against four triangles simultaneously. The result is the addition of a layer of low-level parallel processing to the high-level multi-core layer.

AN ILLUSTRATIVE EXAMPLE

So how well does all of this parallel processing work? Consider an example burst-point set calculation where the burst-point set is a rectangle of burst points located at a fixed height-of-burst (HOB) relative to the target model (a typical simulation that might be run by legacy endgame codes).

The exercise was performed on a typical laptop computer with an Intel i7 four-core CPU. The target is an ground vehicle model that has a set of vulnerable components (occupants, hydraulic lines, wire bundles, engine controls, etc.) typical of industry and DoD standard target geometric models (TGMs).

The warhead ejects 2,000 fragments in a 20-degree side spray. Fragment mass is 240 grains each, and fragment ejection speed is 6,000 fps. The missile carrying the warhead is approaching in anti-parallel fashion (head-on approach direction). Warhead burst points are spaced 1 m apart in a rectangle measuring 10 m by 10 m. The HOB is 5 m above the target geometric center. At each burst location, 20 Monte Carlo point-burst calculations are performed, each of which results in a PK value for the target of between 0 and 1 as a statistical indication of the likelihood of destroying or disabling the target in such a manner that it cannot perform its intended mission.

Averaging the 20 individual point-burst PK values yields a single-shot-average-PK value for the burst point. Figure 3 illustrates the field of burst-point markers color-coded by PK (blue equaling a probability of 0 and red equaling 1).

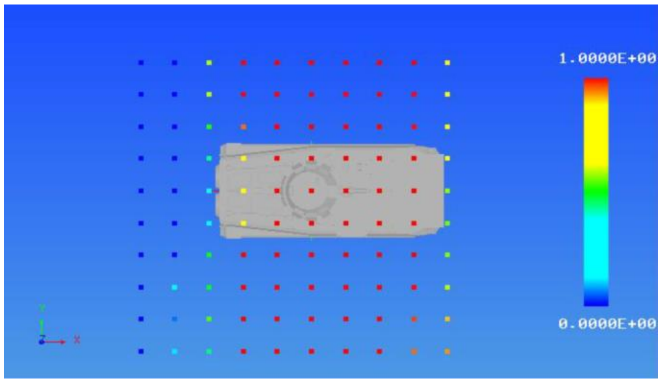


Figure 3: Burst-point field PK values.

There were 100 burst locations in this calculation, so there was a total of 20,000 point-burst calculations. Each point-burst calculation involved 2,000 randomized fragment shotlines. It also involves a set of 14,400 “blast rays” generated at 3-degree intervals in polar angle and roll angle. Performing 20 Monte Carlo samples at each of 100 burst points therefore involves simulating a total of $3x10^7$ rays. Averaged over all 100 burst points, the PK is 0.7124 for this example with a runtime of 0.353 s.

A higher resolution version of the example calculation is shown in Figure 4. In this case the spacing between burst points was set to 0.1 m, which results in 9,800 burst points. This calculation involves roughly $3x10^9$ fragment rays and required 22.5 s of calculation time.

Figure 5 shows the Windows Task Manager during the 9,800 burst-point run. It shows that TurboPK is using 100% of the CPU computing power. Core i7 processors have four full computing cores, but each core is “hyperthreaded” so two separate computational threads share each core’s computing resources. Hyperthreading

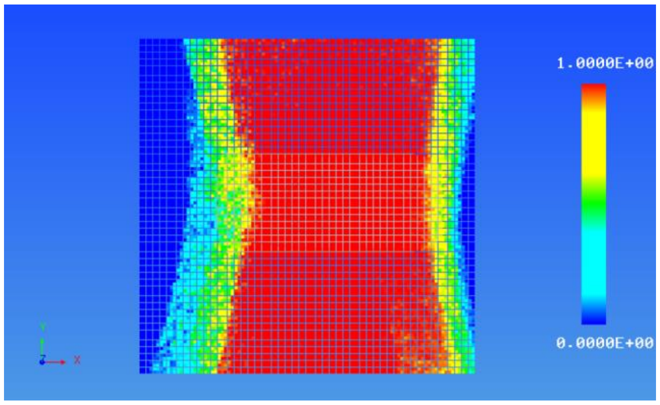
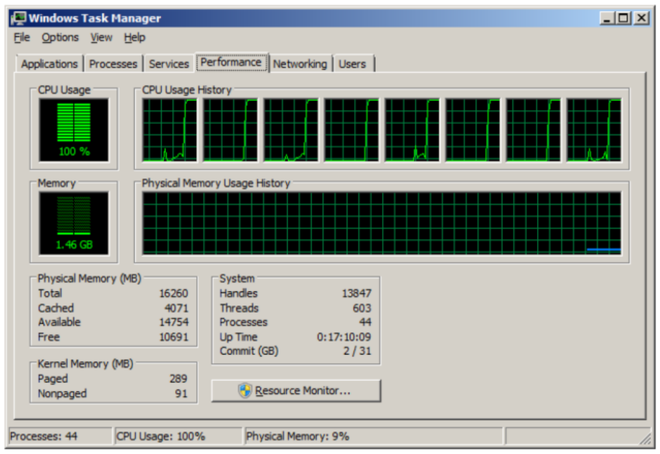


Figure 4: Burst-point spacing reduced to 0.1 m.



Physical Memory (MB)	
Total	16260
Cached	4071
Available	14754
Free	10691

Kernel Memory (MB)	
Paged	289
Nonpaged	91

System	
Handles	13847
Threads	603
Processes	44
Up Time	0:17:10:09
Commit (GB)	2 / 31

Processes: 44 CPU Usage: 100% Physical Memory: 9%

Figure 5: Windows performance monitor.

appears to Windows as eight separate computations running simultaneously, so its performance window reports the usage of eight “CPUs.” TurboPK was built from the ground up to take advantage of multi-core processor designs through parallel programming techniques, and Figure 6 indicates it is successful in using all cores at the same time.

To illustrate the value of parallelizing the calculations, a simulation (describe simulation here) was run on a single core and then on two, three, and four cores (as indicated in Figure 6. Run time for one core was 145 s and for four cores was 41 s, or a speedup factor of 3.54. That result is 88.4% of the theoretical maximum speedup factor of 4.0.

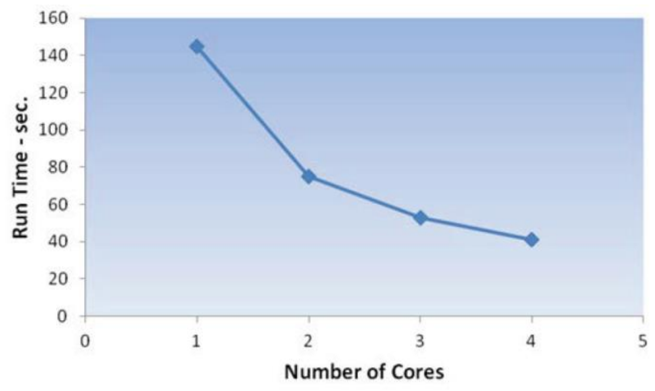


Figure 6: Simulation results for different numbers of CPU cores.

Similar results have been demonstrated for other burst-point set problems with different warheads types and configurations and different targets, including various ground vehicles, aircraft, and personnel. So it is safe to say that parallelizing endgame codes through multi-core computing is well worth the extra programming effort required.

REFERENCES

[1] Intel Corporation. "Embree High Performance Ray Tracing Kernel." http://embree.github.io/, accessed August 2015.