Process Scheduling in Parallel Systems for Ground Vehicle Simulations

Cesar Lucas[1], **Jacob Brendle**[1], **and Jeremy Mange**[1]
[1]Analytical Development and Data Science Team, US Army - TARDEC, Warren, MI, USA

Abstract—*Modeling and simulation (M&S) is an important component of the analysis and design of ground vehicle systems, and parallel computational resources are key enablers of M&S technologies, particularly at large scales. In order to use parallel resources effectively, however, intelligent process scheduling algorithms are needed. In this paper, we first provide a brief overview of ground vehicle modeling and simulation within our research context and the high-performance computing environment we utilize, then define several process scheduling algorithms for these types of problems, and finally compare these algorithms on a variety of problem instances. This research forms a part of an ongoing research project, and the conclusions about which scheduling algorithms are most promising for efficient allocation of parallel resources are an important step forward for enabling continued discoveries within this domain.*

Keywords: Modeling & Simulation, Parallel Scheduling, Ground Vehicles

1. Introduction

As computational power and sophistication have grown, the modeling and simulation of physical phenomena has become an increasingly accurate method of gaining insights into various types of systems without the effort and expense of physical testing. For us as researchers within the military ground vehicle community, modeling and simulation of ground vehicles in particular is a key methodology both for maintenance and improvement of existing vehicles and the design and fabrication of new ones.

However, as the complexity of computer models grows, so too does the computational power needed to simulate them accurately. Parallel computer resources, and in particular high-performance computing (HPC) can greatly aid in producing massive amounts of computational work in reasonable timeframes. This in turn requires intelligent and efficient scheduling in order to take advantage of these resources.

In this paper, we compare several algorithms for scheduling ground vehicle simulation processes within a parallel system. The paper is organized as follows: section 2 provides an overview of ground vehicle simulation in the context under consideration, particularly highlighting why these types of simulations produce certain challenges for efficient parallel scheduling; section 3 describes the HPC environment in which the jobs will be scheduled, as well as the use and limitations of the scheduler itself; section 4 describes the scheduling algorithms to be compared and the experimental design for the comparison; section 5 provides the results of the comparison; and section 6 contains some conclusions about those results, as well as our future plans to continue and expand the project of which this paper is a part.

2. Ground Vehicle Simulation

For our simulation of ground vehicles, we have created a physics-based simulation software that makes use of the main components of certain other simulation packages to perform its own co-simulation. The open source package Chrono handles processing tasks related to the main vehicle body dynamics [2]. A government-developed software package, GCE (Ground Contact Element), handles the ground-terrain interaction [4], and an academic set of tools, PACE (Powertrain Analysis Computational Environment) simulates the powertrain components [3]. All of these are coordinated through a central co-simulation framework which facilitates data passing, time coordination and synchronization, and pre- and post-processing.

The vehicle body dynamics software uses links and relationships between sets of components to model the vehicle. The number of links depends on many things; first, it depends on the type of vehicle. For example, different vehicles have different numbers of axles, and within our framework, each of these axles has one of six different modeled suspension types. Each of the different suspension types has a unique number of bodies and links of which it is composed. In addition, each axle could have an anti-roll bar (ARB), which, although only composed of one additional body, adds a number of additional constraints that have to be factored into the calculations done on the vehicle. In addition, any axle could be steered. While traditionally most vehicles are only have a steering subsystem on the front axle, there are a few vehicles that we deal with that have multiple steering subsystems. Apart from the axles, each vehicle has its sprung mass defined in some fashion. Some vehicles have one generalized sprung mass point, while others very specifically define every location of sprung mass that is on the vehicle. Obviously, the vehicles that have multiple definitions require significantly greater computational cost.

Because of all of these factors, it can be very hard to accurately estimate the length of the simulations, which was part of the motivation for this work – we need an algorithm that efficiently schedules processes for which the computational lengths are difficult to accurately estimate.

Other factors play into the uncertainty of the length of our simulations. For example, the interaction between the vehicle and the terrain requires a large number of computations. These computations vary significantly based on the size of the contact area between the vehicle and the ground. For example, the wheeled vehicles that we generally deal with have anywhere from four to eight tires. An eight wheeled vehicle will require much more processing power because there will be a large number of contact points in comparison to a four wheeled vehicle. Tracked vehicles require the greatest amount of computational time to run out of any of the vehicles we deal with, due to their very large ground contact area. The large variance of contact area between the vehicle and the terrain, as well as the multiple different approaches to modeling the relationships between the contact patches and the terrain itself are yet more factors that add to the difficulty of accurately predicting simulation lengths.

3. HPC Environment

The DoD Supercomputing Resource Center (DSRC) – the home of the main parallel resources we utilize – is a multi-location, high-performance environment that provides multi-processor computing capabilities for Science and Technology (S&T) and Test and Evaluation (T&E) purposes. These resources are part of the DoD High Performance Computing Modernization Program (HPCMP). The DSRCs allow users to offload a significant amount of computing work, freeing the user to continue their workflow without interruption.

Some DSRC systems utilize a Massively Parallel Processing (MPP) method, in which sets of processors are divided with independent buses and memory modules. The network is the critical path used to communicate data among nodes. Therefore, scaling can be achieved simply by adding additional hardware on the network fabric as needed. This allows greater scaling performance in jobs that are designed to be run in parallel, while maintaining independent computation ability. Whereas, Symmetric Multi-Processing (SMP) methods, where a single large job that is highly dependent on shared memory among all processors, is ideal for quick data transfer along the shared memory bus. However, the system's hardware limits scaling in SMP jobs, as one can only add processors and memory within the context of the system. In our context, MPP methods are used.

A single job is typically submitted into a submission queue, where the batch system considers jobs based on a number of factors. Some of these factors include estimated CPU time, memory usage, number of processors to be used, as well as number of nodes to be included in the job. Varying configurations can be considered to optimize CPU utilization, memory usage, and job completion rates. Normally, estimated job time is a primary consideration, which is explicitly stated within the job submission script. In a typical simulation run, our vehicle simulation job is pre-assigned a number of processors and a finite amount of processing time to complete. This number of processors is distributed equally among the number of jobs that is provided, while the amount of time to complete is typically overestimated, due to a hard cutoff at the end of the requested time. This overestimation allows ample time for clean-up and memory recovery. If jobs are forced to run within the confines of a limited number of processors, then context switching among processes would occur between two jobs, causing a significant decrease in performance.

For our purposes in this paper, we want to compare scheduling algorithms and ascertain whether the resulting schedule has a certain "goodness" value. This score can be calculated independently from the number of processors a system has available. By having this calculated goodness score, some jobs are submitted before others, freeing up resources to launch additional jobs without the need of context switching. The simplest metric that captures all of these important factors is average core efficiency; that is, for each core, the average percentage of the time during the job allocation that is spent doing computational work, as opposed to sitting idle. This metric is somewhat independent of the number of cores requested, but capture the efficiency effects that we are most interested in for our comparison of scheduling algorithms.

4. Scheduling Algorithms

Many process scheduling algorithms within parallel environments have been proposed, implemented, and studied ([5] [1]), ranging from fairly simple to very nuanced and complex. Several somewhat unique factors about both the parallel HPC environment and the types of simulations we generally use affected our choices of which algorithms to compare.

In terms of algorithms used to schedule processes on the HPC cores, our first consideration was the number of cores to request for each submission. For the purposes of this comparison, we split these submissions into two categories: a "full" core request, in which a full number of cores matching the number of processes is requested, and a "partial" core request, in which fewer cores are requested than the number of processes to be run. Note that in only the second case is the actual scheduling algorithm relevant, since for a full core request, each process can be assigned to a separate core. Even in this case, however, the core efficiency metric discussed previously is a relevant measure.

For partial core requests, we considered three common and relatively straightforward scheduling algorithms: naive scheduling, shortest-process-first scheduling, and longest-process-first scheduling. For each of these algorithms, a list of processes is maintained, and a central controller assigns these processes to cores, according to the following rules for each scheduling type.

1) **Naive scheduling**: the simplest approach – whenever a core is available, a random process is assigned to it.
2) **Shortest-process-first scheduling**: whenever a core is available, the process with the shortest estimated computation time is assigned to it.
3) **Longest-process-first scheduling**: the inverse; whenever a core is available, the process with the longest estimated computation time is assigned to it.

5. Results

For each set of experiments, we compared jobs composed of various types of vehicle simulations: a homogeneous job submission, in which all of the vehicle models were the same, and the same mobility test was performed repeatedly; a job submission in which the same test was performed but a variety of vehicle models were used; a job submission in which the same vehicle model was used but a varied of tests were formed; and a job submission composed of both various vehicle models and various mobility tests. Therefore, the graphs in this section each has four distinct lines, one corresponding to each of these job submission types.

The first set of experiments we performed involved a "full" core request – that is, an HPC job submission in which the number of cores requested exactly matched the number of processes to be run. We ran these tests over a range of values for the number of processes, ranging from 1 to 1000. The results are shown in Figure 1.

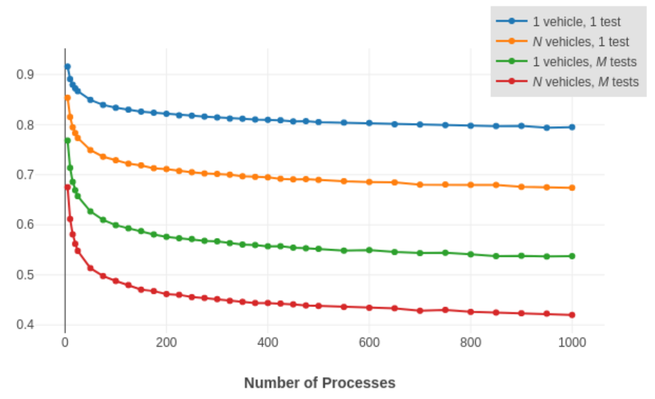

Fig. 1: Full Core Request.

These results are not particularly surprising, but still somewhat illuminating. One can see a clear distinction between the average core efficiencies of the different job compositions, as well as a decreasing core efficiency as the number of processes increases. This effect is most pronounced with a relatively small number of processes, and becomes much less significant with a larger number.

The next set of experiments involved naive, or random, process selection, in which a random process is assigned to a core as available. Again, core efficiency is the main metric of interest, but in this case we varied the number of cores requested as a function of the number of processes, from 10% to 90%. The results for this scheduling algorithm are shown in Figure 2.

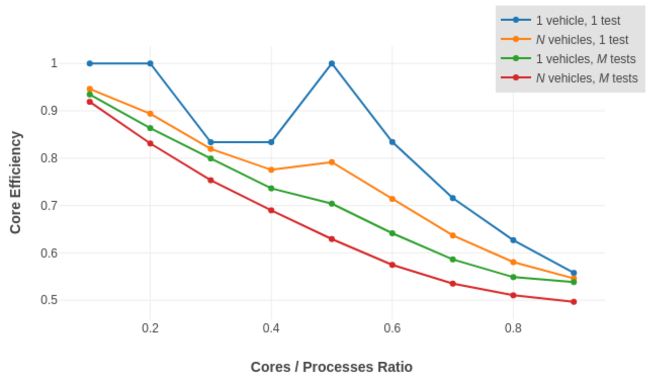

Fig. 2: Random Process Selection.

Interestingly, while the core efficiency shows a clear downward trend as the ratio of requested cores to number of processes increases, there is a somewhat significant increase at the 50% point. This anomaly can be seen in each of the following graphs and appears to be an artifact of the algorithms ultimately effectively assigning exactly two processes to each core in succession, with an average result of core efficiency that is somewhat higher than might otherwise be expected in the overall trend. This is a very helpful observation that can be incorporated into the projectâĂŹs future scheduling and job submission core requests.

Our third set of experiments involved shortest-process-first scheduling, in which the process with the shortest estimated run time is assigned to a core as available. The same metric and experiments were used as in the previous set of experiments, and the results for this algorithm are shown in Figure 3.

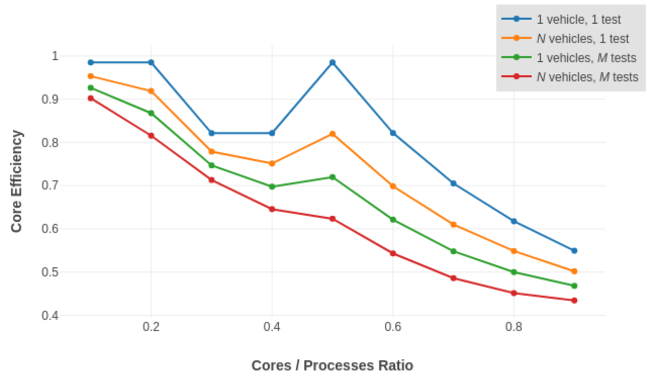

Fig. 3: Shortest Process First.

To our surprise, these results are very similar to those from the random process selection, with, in fact, a slight decrease in average core efficiency for each of the job compositions. This was contrary to our expectations, and gives some evidence that the shortest-process-first algorithm is a choice particularly ill-suited to our particular problem context.

Finally, our last set of experiments involved longest-process-first scheduling, in which the process with the longest estimated run time is assigned to a core as available. Again the same metric and experiments were used, and the results for this algorithm are shown in Figure 4.

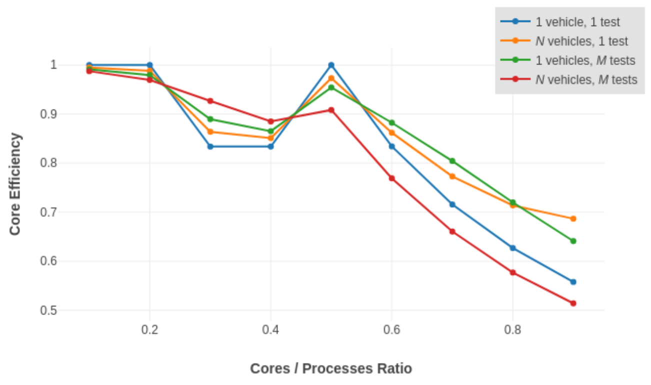

Fig. 4: Longest Process First.

These results are much more promising, and show a significant improvement in average core efficiency for each of the job compositions, particularly at smaller ratios of requested cores to number of processes. Interestingly, the ordering of job compositions by core efficiency completely changes at certain ratios, instead of maintaining the same ordering as in the other experiments. This lends support to the conclusion that not only is longest-process-first scheduling the most promising in terms of overall core efficiency, but it also appears the most robust, as it produces a more efficient schedule for jobs even with heterogeneous compositions, which is a common scenario within our problem context.

6. Conclusion and Future Work

A wide variety of process scheduling algorithms exist, and many have been studied and compared within various problem contexts. The three we chose to compare in this paper have many variations and derived sub-algorithms, which was part of our motivation for choosing them – to see which "family" of algorithms showed the most promise for assigning ground vehicle simulation jobs to high-performance computing cores most efficiently.

Based on the results from these experiments, the longest-process-first scheduling algorithm is by far the most effective for the types of ground vehicle modeling and simulation we tested. This is a very helpful discovery for our project, and will shape our approach to parallel process scheduling for the future.

As this project continues, we plan to implement and test several other more complex process scheduling algorithms, including variations of the longest-process-first algorithm that performed so well in these tests. We also plan to continue using the parallel resources described to perform a variety of ground vehicle system modeling and simulation tasks, to improve current capabilities and help inform future vehicle designs.

References

[1] Andrea C Dusseau, Remzi H Arpaci, and David E Culler. Effective distributed scheduling of parallel workloads. *ACM SIGMETRICS Performance Evaluation Review*, 24(1):25–36, 1996.

[2] Hammad Mazhar, Toby Heyn, Arman Pazouki, Daniel Melanz, Andrew Seidl, Aaron Bartholomew, Alessandro Tasora, and Dan Negrut. Chrono: a parallel multi-physics library for rigid-body, flexible-body, and fluid dynamics. *Mechanical Sciences*, 4(1):49–64, 2013.

[3] J Monroe, Matthew Doude, Tomasz Haupt, Gregory Henley, Angela Card, Michael Mazzola, Scott Shurin, and Christopher Goodin. Thermal modeling in the powertrain analysis and computational environment (pace). *NATO AVT-265*, 2017.

[4] Paul W Richmond, Randolph A Jones, Daniel C Creighton, and Richard B Ahlvin. Estimating off-road ground contact forces for a real time motion simulator. Technical report, SAE Technical Paper, 2004.

[5] Behrooz A Shirazi, Krishna M Kavi, and Ali R Hurson. *Scheduling and load balancing in parallel and distributed systems*. IEEE Computer Society Press, 1995.