Project Chrono

Onyx Bennett, Brennan Krutis, Josh Derrow

Our project lives in /scratch/chrono on the cluster.

Background:

PROJECTCHRONO is a multi-physics modelling and simulation infrastructure based on a platform-independent, open-source design. The core of PROJECTCHRONO is the Chrono::Engine middleware, a C++ object-oriented library that can perform multi-physics simulations, including multibody and finite element analysis. This software is used by researchers, industry professionals, and government agencies to simulate real-life scenarios (NASA Rovers, Shear Displacement). It is primarily maintained by the University of Wisconsin-Madison.

The first version of the Chrono::Engine was developed in 1998 by Prof. Alessandro Tasora when he was a student at the Politecnico di Milano. It was the result of a thesis in Mechanical Engineering. Originally, Chrono::Engine was meant to be a multibody simulation tool for robotics and biomechanics applications.

Today, Chrono has a wide variety of features, notably vehicle modeling, granular dynamics, and fluid-solid interaction. It uses a modular architecture, allowing users to pick and choose which elements of Chrono will be included in their build. One module, ChronoMulticore, enables the use of parallel computing with technologies such as pthreads, OpenMP, MPI, and CUDA.

Building and Testing:

To build ProjectChrono on the cluster, we had to clone the Chrono GitHub repository, create a build folder, run 'cmake', and finally run 'make'. We encountered many issues while building the project. First, we could not build Chrono on the cluster because multiple required libraries, thrust and Eigen3, are provided on the cluster but were not visible to the CMake instructions. We created a bash executable file that sets many environment variables to their correct locations on the cluster before running CMake, which fixed this issue. Once that was addressed, we could build Chrono on the cluster and run a portion of the provided demos. However, this base build of Chrono did not give us access to parallelism; Chrono has many optional modules that can be enabled during CMake with variables. We manually enabled Chrono::GPU (uses CUDA) and Chrono::Multicore (uses pthreads) in our bash file to take advantage of parallelism and analyze the results. This caused one final dependency issue, as ChronoMulticore required a library called ‘Blaze’. This is not included in the cluster, and we had to manually install Blaze into our scratch folder. After we successfully built Chrono with our chosen modules, we analyzed the given demos that come with these modules. We determined that it would be better to work with the ChronoMulticore module when creating our demonstrations.

Since Chrono was designed to be used with third-party visualization software, all of the provided demos contained a lot of code specific to the visual aspects. We could not take advantage of these features on the cluster, and thus only served to take up time. The provided demos also lacked flexibility in changing elements such as thread count, which was the focus of our testing. So, we opted to create custom demos. We decided to create a simple simulation of balls falling that takes input for thread count, ball count, and simulation duration. This allowed enough calculations for us to test strong and weak scaling without excessive run times.

For strong scaling, we kept the ball count at 1000 and the duration at 2 seconds. We used thread counts of 1, 2, 4, 8, 16, 32, and 64. We did get an error message with 64 threads that read “WARNING! Requested number of threads (64) larger than maximum available (32)”. The tests that used 64 threads still ran despite the warning, although it ran at similar speeds, if not slower, than tests using 8 threads. We assume this is why the thread maximum is coded into Chrono, however, we were unable to discern why this happens. For weak scaling, we kept the thread count at 2 and the duration at 2 seconds. We used ball counts of 500, 1000, 2000, and 4000. We also ran tests that kept threads and balls consistent at 2 and 1000, respectively, and changed the duration (.5s, 1s, 2s, 4s, and 8s) just to see if anything interesting would happen.

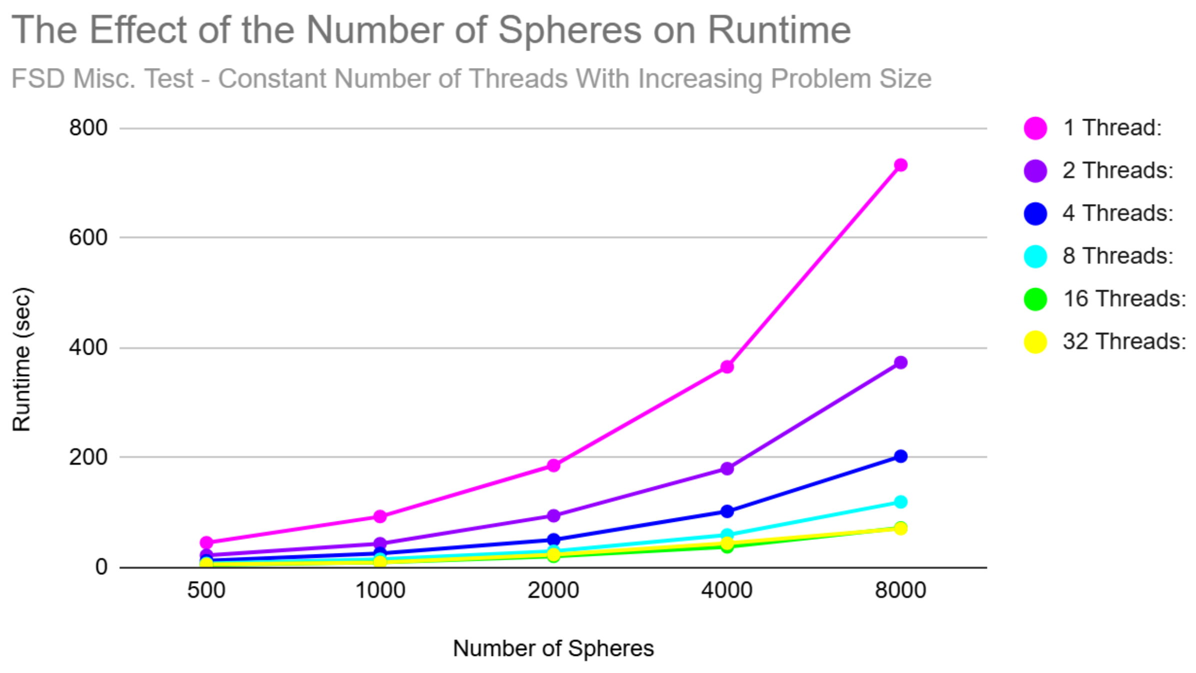
We also attempted to create a demo where balls fell onto a vertical wall so we could see how collisions impacted the number of calculations. We hoped there would be a change in scaling, since there would be potential for some threads to take on more work than others, depending on whether some balls had more collisions than others. However, when running similar scaling tests as mentioned above, the run times ended up being the same as the non-collision ball demo. Without visualization, we were unable to determine if these results were correct or if there was an issue, such as not running the program long enough or a misalignment where the balls should have hit the wall.

In hindsight, we believe we would have been more successful in our testing if we had narrowed our focus to a specific aspect of Chrono or picked a system that was either smaller or less reliant on visuals. Much of our time was spent figuring out how to use Chrono rather than actually using it.

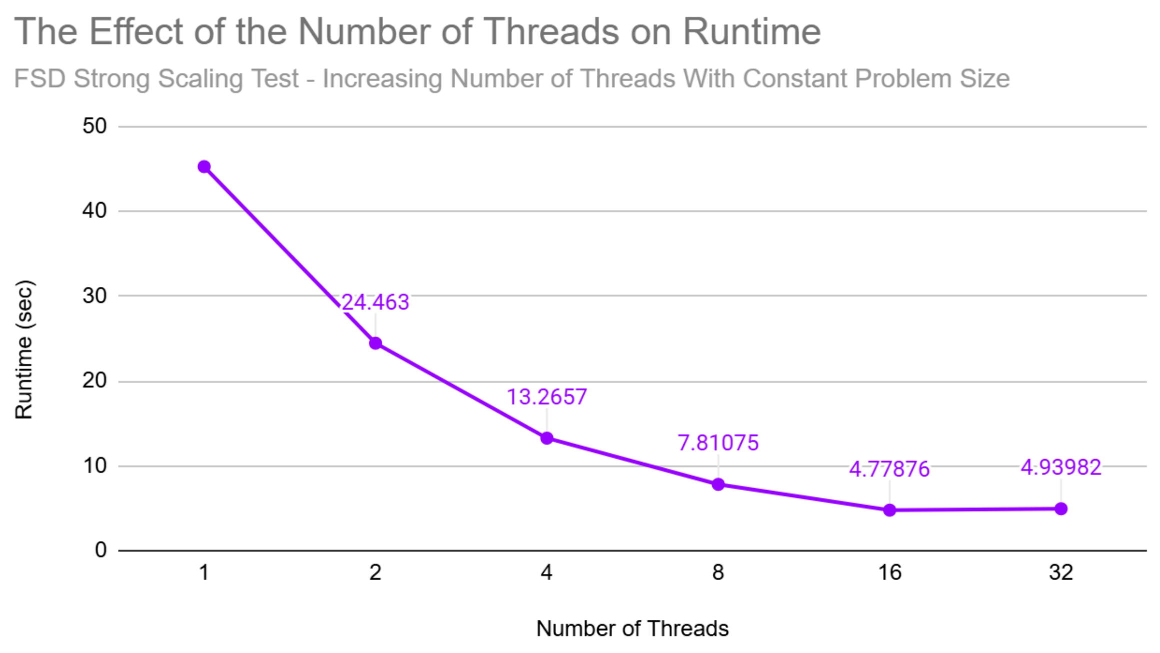
Analysis and Results:

For our analysis of this system, we used our custom demonstration mentioned above (path is /scratch/chrono/demo_files). We initially wanted to analyze the performance of the built-in demos of the multicore module; however, since this system is primarily used as middleware to be visualized by other software, we were unable to properly test the built-in demos. Our demo, ballDemo.cpp, creates a provided number of spheres and simulates gravity's effect on them over a duration of time. The Multicore module includes a setNumThreads function, which we used to manipulate the number of pthreads.

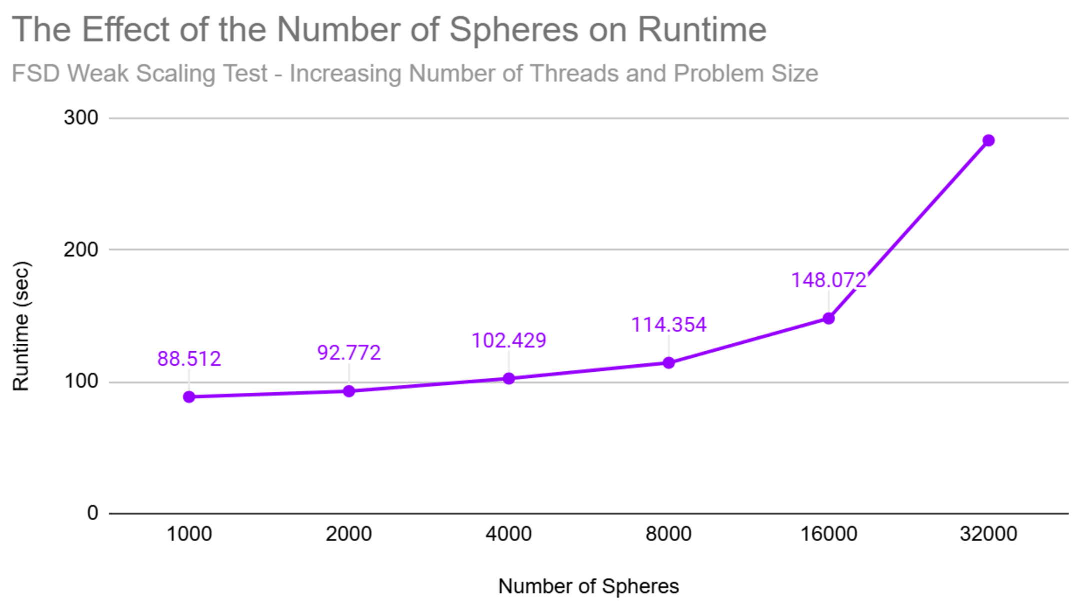
We constructed a bash executable file to run a series of tests to analyze the performance (path is /scratch/chrono/sh_files/runBallDemoTesting.sh). The graph below shows the results of running the demo with 500, 1000, 2000, 4000, and 8000 balls. We ran this for thread counts of 1, 2, 4, 8, 16, 32, and 64.



This graph shows us that this aspect of Chrono is naturally parallel and scales strongly. As input size doubled, run time also doubled. As thread count doubled, run time halved. Our next test was keeping the problem size constant (1000 spheres) while doubling the number of threads. (Data below, duration: 2)

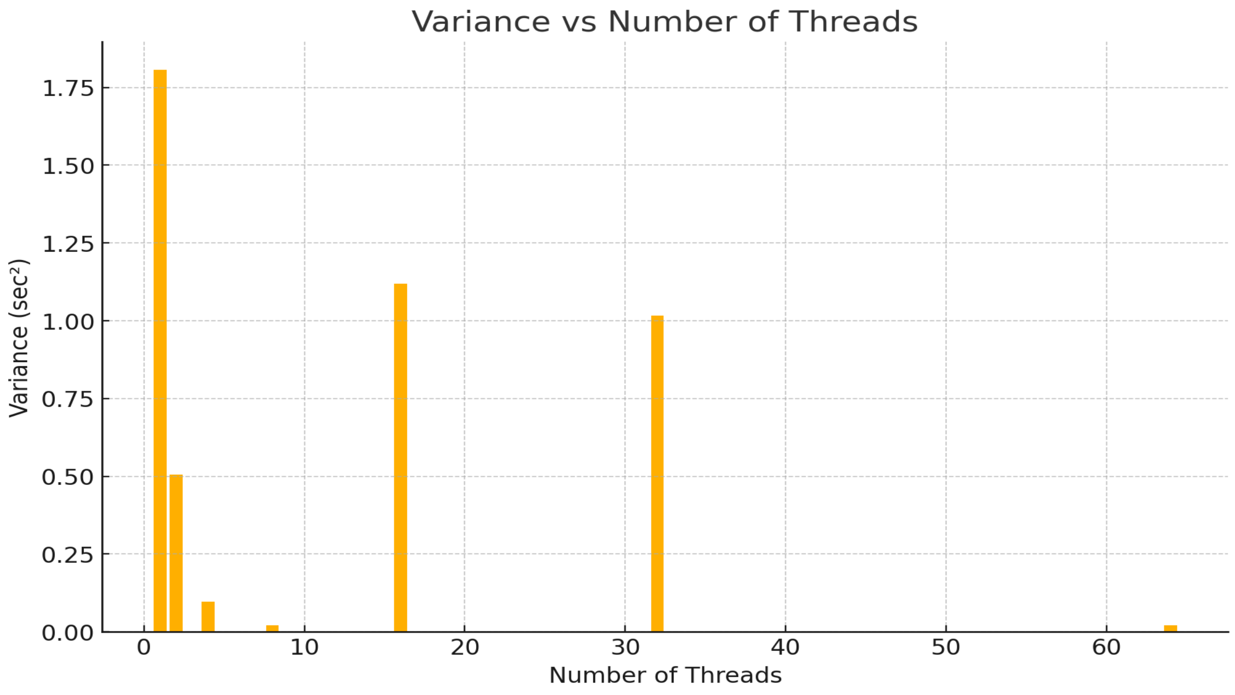


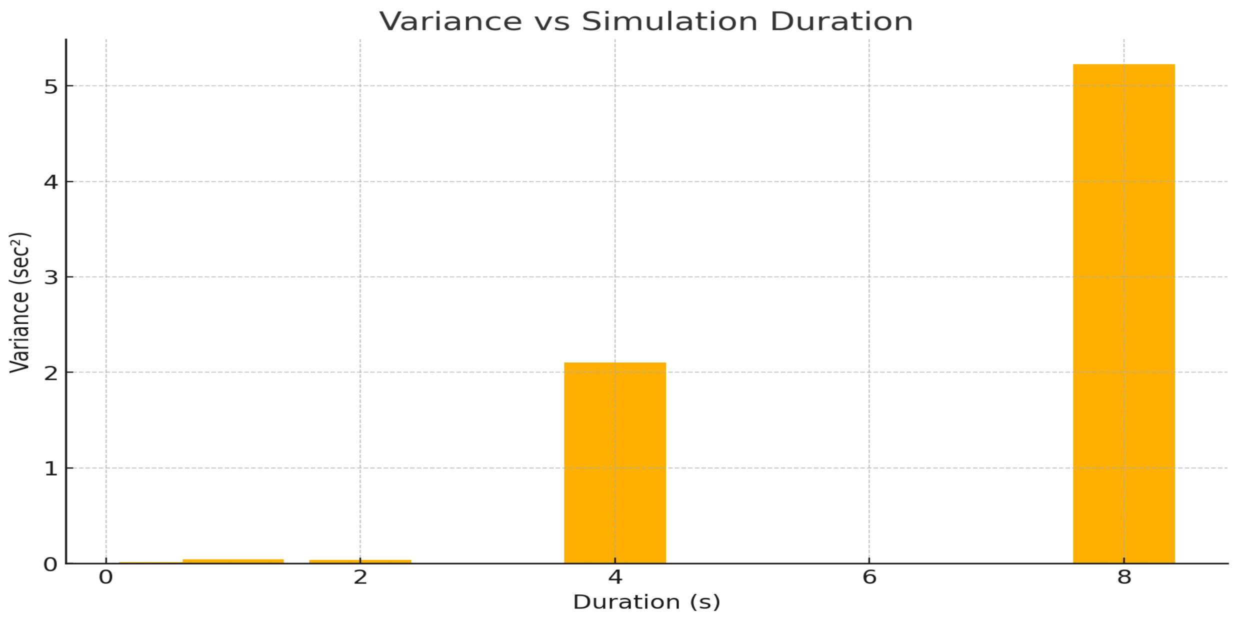
These results indicate that this system is strongly scaling, as each time the number of threads is doubled, the runtime of the problem roughly halves. Something of note is the slight increase in runtime on 32 threads. Over repeated testing, this has been rather consistent behavior, likely indicating a lack of hyperthreading support. The last test we conducted was doubling both the problem size and the number of threads, while keeping the duration constant. (Data below)



From the results of the graph, we concluded that this system is also weak scaling, as the runtime stays roughly the same as we double both the thread count and problem size. Something interesting to note is that the system seems to stop weak scaling when the thread count/sphere count reaches 32/32000. This likely indicates that the underlying implementation of parallelism in the multicore module does not take advantage of hyperthreading, as our cluster nodes have 16-core CPUs.

We ran each test 5 times to calculate the variance in time between runs:





These graphs show that as the thread count increases, the variance of runs lowers, and that as the simulation duration increases, there is a noticeable increase in variance.

Future Work:

This system has been regularly maintained by members of the Simulation-Based Engineering Lab at the University of Wisconsin-Madison since 2007, and there is an active Google Groups forum for Chrono that has many new questions or bug notifications, with quick replies from experts. This project is also open source, which allows anyone to contribute their solutions to these bugs. Over the years, there have been over 75 contributors to the project on GitHub. Commits are being pushed almost every other day, whether they be bug fixes, new implementations, or optimizations of previous components. If anyone were to expand on our findings of Chrono, we would suggest looking into why Chrono is unable to handle more than 32 threads. There is potential for optimization using pthreads, and it may also be worth finding a way to use OpenMP instead of pthreads.