Team: Josh Derrow, Brennan Krutis, & Onyx Bennett

Software System: Project Chrono - https://github.com/projectchrono/chrono

For our project, we are working with Project Chrono (initially released in 1998, main GitHub branch was just updated today 3/7/25), which is an open-source package for physics models, simulations, and visualizations. Project Chrono has models that support rigid & flexible multibody dynamics, granular dynamics, fluid-solid interactions, vehicle & robotic simulations, and terramechanics. With so many real-world applications, Project Chrono is a useful tool for researchers, industry professionals, and the federal government.

Technologies:

This project uses many different technologies and is very configurable. Therefore, it employs the use of many optional libraries, while also having one mandatory dependency. The core dependency of Project Chrono is the Eigen3 linear algebra library, Chrono cannot be built without it. There are numerous other optional dependencies including but not limited to: BLAS, Blaze, IrrKlang, Irrlicht, OptiX, Spectra, & Thrust. This project also supports the use of many of the topics we have discussed in class, such as Pthreads, OpenMP, MPI, and CUDA. Since some parts of the code run on distributed memory architectures via MPI, Project Chrono is also a distributed system.

The vast majority of this project is written in C++ (83.4%) but it also contains code written in C (7%), Cuda (2.9%), Python (2.2%), CMake (2.2% - What the build system is based on), SWIG (1.1%), & others such as C#. The size of the project by itself is 1.9 GB and the size of our current build is 1.7 GB, which yields a total size of 3.6 GB. In total, there are approximately 628,188 lines of code in our project and approximately 12,243,037 lines of text. This is quite a lot and shows the age, scale, and configurability of Project Chrono.

Progress:

So far, we have been able to compile and build Project Chrono on the cluster successfully, and we have run many of the provided demos. On top of the basic build, we have also installed various modules that Chrono has to offer, including Chrono::GPU, Chrono::SYNCHRONO (Uses MPI), Chrono::Vehicle, and Chrono::Multicore, each of which provides multiple demonstrations of their use cases. While many of these demos require HPC Visualization software, others show their computations through the CLI. It can be observed that the hardware is being utilized on the GPUs properly, as when srun is run without the GPU node requested, demos that require CUDA will not run, but works when "--gres=gpu" is included.

We are also currently in the process of verifying the availability of the optional dependencies on the cluster. Some of them may already be installed like Eigen3, while other dependencies may not already be present/able to be installed on the cluster. We have not done any performance experiments yet, but that will be one of our next steps after we run more demos and get fully settled with the configuration of our project build.

Roadblocks:

We have run into a few obstacles so far with Project Chrono, but nothing major that can't be overcome. First, we could not build Chrono on the cluster because the Eigen3 library was not visible and we didn't know it was already on the cluster. It was already on the cluster though, so we fixed this issue by specifying the environment variable Eigen3_ROOT with the correct path in the build command (Eigen3_ROOT="/shared/common/eigen-3.4.0/").

Once the correct path was specified in the build command, we were then able to build Chrono on the cluster and run a portion of the provided demos, but not all of them. When running the available demos, we noticed that they were producing the same output regardless of what cluster node we were running the demos on (login, compute, & GPU). This led us to our next issue: the demos were not utilizing the nodes properly, especially the GPU nodes. We later found that the cause of this issue was that the GPU module had to be loaded by changing a CMake option.

Another roadblock that we encountered was when trying to enable the Chrono::OpenGL module. This module requires Blaze, a high-performance C++ math library, and we weren't able to find a directory on the cluster that contained this library.

Moving forward, we are still unable to graphically render any of these models without visualization software (not currently set up on the cluster). However, we can still run the demos and perform all of the necessary computations. Here is a link to a playlist from one of the project developers of the various GPU simulations visualized with ParaView ($10,000 per installation): https://www.youtube.com/watch?v=iSxYApFgqD0&list=PL1uQAy4kmxZBE7SJPRMakG_js3FPnDgFd

To potentially work around this, we are looking into extracting the computed information from the cluster and running the data on one of our local machines, which can support a wider range of HPC visualization software.

Insights and Contributions:

We haven't made any contributions to date. The GitHub for Chrono currently has 96 open issues. Ideally, we would like to find a significant bug we are capable of fixing, or at least a few small ones. This goal may be unattainable, as looking through the reported bugs we have not found one that we are confident we could resolve. As for other insights, Project Chrono has a community board (available here: https://groups.google.com/g/projectchrono) that serves to "exchange ideas, comments, and bugs with other developers of Project Chrono". We are using this resource as a guide to find an area where we can contribute. Currently, one of the main issues people seem to have with Chrono is installation. There is an installation guide on the ProjectChrono website that we could revamp to include solutions to common problems that people encounter. Users report a large learning curve in getting started with Chrono, so in addition to an improved installation guide, we could create documentation to help users run the demos and create starter projects of their own.

Expanding on documentation, Chrono is frequently used with other software as well as subsections of itself. We think it would be helpful to create guides on how some of the most common companions interoperate with Chrono. Additionally, some parts of Chrono are not fully compatible with each other. For instance, ChLoaderGravity cannot be used with ChBody. A comprehensive list of 'what works with what' would save users a lot of headaches.

There is also the potential for us to create compatibility. In response to the ChLoaderGravity-ChBody issue, a developer stated

"ChLoaderGravity was meant for ChFiniteElement objects, to automate some computations inside ChMesh container, but was NOT meant for ChBody. In theory it could be extended to be compatible with them, but it would require some workaround because bodies do not have a xyz domain for Gauss quadrature, and their mass comes precomputed by the user. "
(https://groups.google.com/u/4/g/projectchrono/c/NIDGUN1OXvA/m/wv9vO_T6BQAJ)
If possible, we could add the necessary code to allow these features to be compatible.

For another more technological contribution, a user recently found some inefficiencies in the arithmetic throughout the code (seen here: https://github.com/projectchrono/chrono/issues/550). We could follow up on this user's request, finding computations that have faster alternatives and replacing them while ensuring accuracy. For example, Chrono often uses **std::pow(x,3)** instead of **x*x*x** (which is the more readable option in my opinion). The user who reported the issue found that making this change made a significant impact on performance. We would, of course, do our own testing. There are other general complaints from users about simulation speeds. These issues are broad, but we believe we could find specific areas of code where we could either optimize existing threads or add them to otherwise serial blocks.