Simulation-Based Engineering Lab
University of Wisconsin-Madison

Technical Report TR–2017–04

Hybrid OpenMP - MPI Simulation for Large-Scale Granular Dynamics in Chrono

Nicholas Olsen, Radu Serban, and Dan Negrut

Dept. of Mechanical Engineeering, University of Wisconsin – Madison, Madison, WI

January 4, 2018

Abstract

This technical report details a new hybrid OpenMP - MPI infrastructure for performing large-scale granular simulation in Chrono. This hybrid support is provided in a new module, Chrono::Distributed, which builds on the existing OpenMP-enabled module, Chrono::Parallel.

Keywords: MPI, OpenMP, Granular Dynamics

Contents

1 Introduction

1.1 Previous Support

Chrono is an open-source, BSD3 available, multi-physics simulation engine developed through a joint university effort [4]. Until recently, the highest level of support for simulating granular dynamics in Chrono has been provided by Chrono::Parallel, a module which uses OpenMP acceleration to expedite computation of large many-body systems. This implementation is limited by (i) cache coherency issues that prevent Chrono::Parallel from achieving a more than 15-fold speedup despite using workstations that sport 64 parallel cores; and, (ii) constraints on memory size available on a workstation.

1.2 Necessity of a Hybrid Implementation

Depending on grain size, one cubic meter of sand contains well over a billion elements. This problem size renders futile any Chrono::Parallel attempt for a fully resolved analysis of vehicle mobility on sand, the drifting of a sand dune, etc. As pointed out in [3], more than 50% of the materials processed in industry come in granular form and understanding their dynamics is relevant in a range of practical applications such as additive manufacturing, terramechanics, nanoparticle self-assembly, composite materials, pyroclastic flows, formation of asteroids and planets, meteorite cratering; and also in industries such as pharmaceuticals, chemical and biological engineering, food processing, farming, manufacturing, construction, and mining.

Against this backdrop, we have developed a new Chrono module called Chrono::Distributed, which has been designed to harness the compute power and RAM available on multiple nodes to tackle problem sizes that reach into billions. The new module implements a distributed memory, parallel computing framework that uses the Message Passing Interface (MPI) to wrap the existing OpenMP multi-threading implementation in Chrono::Parallel thus allowing for a modular, stacked software deck: Chrono::Distributed leverages Chrono::Parallel, which relies on the Chrono::Engine.

2 Overview of Hybrid OpenMP - MPI Solution

2.1 Numerical Model

The numerical solution methodology adopted formulates the equations governing the time evolution of a mechanical system as a collection of index three differential algebraic equations of motion that account for friction and contact via external forces. The numerical solution of the equations of motion draws on a first order, semi-implicit symplectic Euler method [1]. The current Chrono::Distributed solution models friction and contact using a penalty-based approach. Unlike the differential-variational inequality approach available in Chrono, the penalty approach leads to a decoupled numerical solution that is more amenable to parallel computing via MPI.

2.2 OpenMP Support in Chrono::Parallel

Chrono::Distributed implements a distributed-multicore hybrid solution to leverage parallelism over multiple nodes each with multiple cores. Multicore parallelism is provided by the Chrono::Parallel module via OpenMP. This is obtained by organizing the underlying data in flat structures of arrays (SoA), which optimizes shared memory access and cache usage, and by using OpenMP parallel for loops and algorithms provided by the Thrust parallel library [2] through its OpenMP back-end. Chrono::Parallel implements a custom parallel binning algorithm for its broad-phase collision detection. Resolution of actual collisions is performed in parallel over all candidate pairs in a narrow-phase based on a hybrid analytical - MPR (Minkovski Portal Refinement) algorithm.

2.3 MPI Support in Chrono::Distributed

Support for distributed memory simulation in Chrono::Distributed creates multiple processes, each with an instance of a Chrono::Parallel system. At the onset of the simulation, the user-defined spatial domain is statically divided into sub-domains, each of which is assigned to a single MPI process, or rank. For the duration of the simulation, each rank is responsible for computations only in its designated sub-domain and a thin sharing region between it and the neighboring sub-domains. In these sharing regions, each body is owned by one rank and viewed as a proxy body on the other. Each time step, the pair of ranks use a standard Chrono::Parallel time step separately, viewing their own bodies and proxies from their neighbors. After each time step, each rank sends out MPI point-to-point communications to its neighbors to update the proxy bodies that mirror its owned bodies. Pairs of ranks synchronize body data using non-blocking point-to-point communications to prevent communicator-wide collective communications. This minimizes both the number of messages sent and their sizes in addition to preventing long chains of communication hops within large clusters, largely eliminating the high overhead associated with collective communications over a very large set of ranks. Non-blocking communications allow each rank to overlap its time spent packing data for sends with previous communication times, hiding some of the cost of high-latency MPI communication. In order to maintain a consistent view of the system at all times, each body across the distributed simulation is given a unique global identifier at setup. Each rank then is responsible for mapping each global ID to its own local index for the body. This allows fast lookup of a body across a large distributed system.

3 Scaling Analyses

Strong and weak scaling analyses were performed on a Cray XC30 system on computer nodes with one 12-core Intel® Xeon® E5-2697 v2 processor each and are interconnected with a dedicated Cray Aries high-speed interconnect.

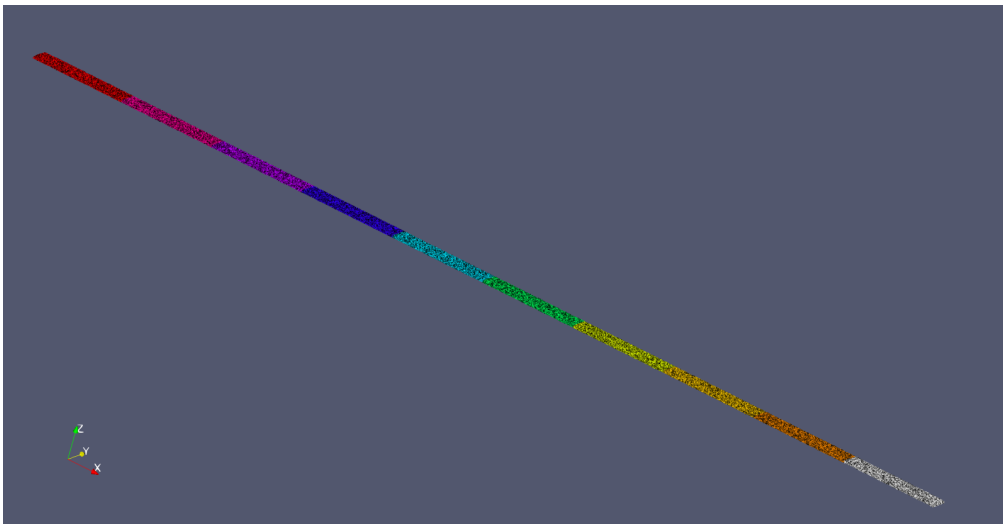

Figure 1: Experimental setup for strong scaling simulates roughly 1 million bodies setting in a long box. Color indicates the rank on which the bodies are handled.

MPI Ranks	Particles	Problem Size Ratio	Wall Time (s)	Parallel Efficiency
1	1,236,372	1	46,978.9	N/A
2	2,472,744	2	47,801.9	0.983
4	4,944,700	4	47,995.1	0.979
8	9,889,400	8	48,198.7	0.975
16	19,778,012	16	48,812.9	0.962
32	39,555,236	32	48,961.5	0.960

Table 1: Weak scaling data run on the Cray XC30 system. Parallel Efficiency: $E(N) = \frac{T(1)}{T(N)}$

MPI Ranks	Particles	Problem Size Ratio	Wall Time (s)	Parallel Efficiency
1	1,236,372	1	46,978.9	N/A
2	1,236,372	1	23,653.3	0.993
4	1,236,372	1	11,908.6	0.986
8	1,236,372	1	5,544.9	1.059
16	1,236,372	1	2,879.0	1.020
32	1,236,372	1	1,424.6	1.031

Table 2: Strong scaling data on the Cray XC30 system. Parallel Efficiency: $E(N) = \frac{T(1)}{T(N)}$

4 Obstacles to the Distributed Framework

A number of common aspects of multibody simulations pose additional challenges in a distributed-memory environment that are otherwise non-problematic in shared-memory environments.

4.1 Large Bodies

In order to effectively divide a large simulation between MPI ranks, the global problem must be divisible into mostly-disjoint sub-problems. In this way, large bodies, which span multiple sub-domains pose a problem to the disjointness criterion. Chrono::Distributed currently only allows for sub-domain-spanning bodies whose motion is not dictated by external forces. Such large bodies must be either fixed or move without regard to interactions with other bodies. This ensures that a large body cannot complicate the point-to-point communication pattern between two neighbor ranks.

Large bodies also present an issue for the collision detection mechanism. The main collision detection algorithm relies on a binning-based broad-phase. A large body which crosses into another sub-domain complicates this broad-phase by extending the axis-aligned bounding box of the sub-domain to include large amounts of space that are handled by another rank. By forcing the inclusion of this space, the large body drastically reduces the impact of the broad-phase by reducing the fraction of the rank's bounding box that is actually occupied by bodies to be handled on that rank. To combat this, simulations should be constructed so that the collision shape associated with a large body is truncated to only include the portion that lies in the relevant sub-domain. Another, easier, solution to this problem is to represent large bodies as custom collision conditions. Chrono supports this solution by providing the user with a base class from which the user can define derived classes that detail how a custom collision object should be handled at each time step. Defining a large body in this way allows it define its own broad-phase and, in doing so, circumvent the main broad-phase.

4.2 Communication Time

MPI communication between compute nodes is orders of magnitude slower than intra-node communication. The amount of time devoted solely MPI communication should therefore be minimized in order to not drastically impact the simulation time.

Most simply, communication time can be reduced by minimizing the amount of data that must be sent between ranks over the network. Chrono::Distributed achieves this by defining custom MPI datatypes that pack data in a way as to minimize padding. This requires that communication be broken into four distinct types: body exchanges, collision shape exchanges, body updates, and body transfers. Body exchanges notify a rank that an external body has entered its sub-domain and add a proxy ghost body to that rank. Following this, collision shape exchanges add the collision models to these proxy ghost bodies. Body

updates synchronize data for bodies within a sharing region. Body transfers notify a rank that a body has passed out of the sharing region and remove its ghost proxy body from that rank.

Ideally, the communication time would coincide with time spent performing calculations on parts of a sub-domain unaffected by boundary regions. This is not possible in Chrono::Distributed, as it must only wrap Chrono::Parallel, which does not allow for the necessary partial solution functionality. Instead, Chrono::Distributed is limited to covering communication time with time spent in preparation of communication. To this end, the communication phases described above are overlapped as much as possible by using multi-threading from OpenMP to allow one thread to perform MPI communications while other threads gather and pack data for each phase of communication.

4.3 Load Balancing

A major performance pitfall for such a distributed framework is load imbalance. Each MPI rank must synchronize with its neighbors before continuing with its computations. In this way, a single rank slowed by overloading can form a bottleneck for the entire simulation. Currently, Chrono::Distributed is aimed at terrain simulation and as such does not yet have support for adaptive load balancing. The responsibility for choosing an optimal domain decomposition currently lies with the user.

5 Profiling

A simple profiling analysis was performed on single and 10-node configurations on the Euler cluster. The test simulation contained 1 million as with above tests. Compute nodes each have two 10-core Intel® Xeon® E5-2640 v4 processors with QDR Infiniband interconnects. Profiling focused on timing the main portions of the simulation algorithm including broad-phase collision detection, narrow-phase collision detection, solver, and inter-node communication. Timing was performed with existing Chrono timing utilities. Table 3 shows the results with and without the presence of inter-node communication for contrast. The data from the 10-node run is also presented in the final column considering only non-communication time in order to clearly compare with the single-node run.

Profiling results show a drastic shift in the time distribution between runs. The portion of compute time spent in broad-phase is reduced by more than 12%, which proportion spent by the solver is increased by roughly 20%. This is due to the slightly non-linear scaling of the broad-phase, which performs better for smaller problem sizes. Finally, the profile also shows that communication does take up almost 6% of the total simulation time. This indicates that further optimizing the communication could amount to significant performance gains.

Event	% Time-1 Rank	% Time-10 Ranks	% Non-Comm Time-10 Ranks
Broad-phase	35.58	21.73	23.07
Narrow-phase	11.32	10.67	11.33
Solver	43.14	61.11	64.88
Communication	N/A	5.81	N/A

Table 3: Profiling results from two runs on Euler.

6 Conclusions and Future Directions

Chrono::Distributed shows efficient scaling, both weak and strong. This scaling shows the potential for practical billion-body simulations on large clusters.

The following are future additions, improvements, and testing planned for Chrono::Distributed.

- 2D and 3D domain decomposition.
- Integration into the existing co-simulation framework in Chrono.
- Adaptive load balancing to prevent bottlenecking due to non-uniform loading.
- Additional collision geometry support.
- Large body support.
- Further optimization of the underlying Chrono::Parallel implementation.
- Scaling analysis on larger numbers of nodes.

References

[1] E. Hairer, C. Lubich, and G. Wanner. *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations*, volume 31. Springer Science & Business Media, 2006.

[2] J. Hoberock and N. Bell. Thrust: A parallel template library, 2010. Version 1.7.0.

[3] Patrick Richard, Mario Nicodemi, Renaud Delannay, Philippe Ribiere, and Daniel Bideau. Slow relaxation and compaction of granular systems. *Nature Materials*, 4(2):121–128, 2005.

[4] A. Tasora, R. Serban, H. Mazhar, A. Pazouki, D. Melanz, J. Fleischmann, M. Taylor, H. Sugiyama, and D. Negrut. Chrono: An open source multi-physics dynamics engine. In T. Kozubek, editor, *High Performance Computing in Science and Engineering – Lecture Notes in Computer Science*, pages 19–49. Springer, 2016.