DARPA Robotics Challenge
OSRF
Gazebo Parallel Physics Report

Open Source Robotics Foundation
170 South Whisman Road
Building D, Suite A
Mountain View, CA 94041
650.450.9681
info@osrfoundation.org

May 29, 2015

Contents

1 Overview

This document details an analysis of physics parallelization using Open Dynamics Engine in Gazebo.

2 Why parallelization of physics is desirable

The complexity of simulated robots, spatial size of environments, and fidelity of sensor simulation all play a role in determining what can be simulated with the constraint of operating at or near real-time. In the context of simulation, real-time performance is achieved when the time required to solve an iteration of simulation equals the simulated

duration of the iteration. When computation time is less than than the simulated time duration, performance is faster than real-time. The opposite is true when computation time is longer than the simulated time duration.

The Gazebo physics update loop is one of the primary consumers of CPU cycles. With a limitation in the speed of algorithms to solve the mathematical problem that represents those physical constraints, parallelization of the physics engine is an option for improving performance, with the goal of running complex robots and environments in real-time.

3 Parallelization strategies

A significant performance bottleneck in the physics update loop is constraint resolution, where constraints are defined by joints and points of contact between two objects. A physics engine, such as the Open Dynamics Engine (ODE), typically solves system constraints using a monolithic algorithm. This architecture is conceptually simple and handles a majority of use cases. However, such an architecture does not scale well as the simulated environment grows in number of models and model complexity.

Two strategies to parallelize physics have been implemented. The first strategy attempts to parallelize simulation of non-interacting entities. Simulated entities are interacting if they are connected by an articulated joint (such as a revolute or universal joint) or are connected via contact. Groups of interacting entities are clustered into "islands" that are mathematically decoupled from each other. Thus each island can be simulated in parallel. After each step, the clustering of islands is recalculated. Due to the architecture of ODE, this strategy can be used with different solvers, such as the iterative projected Gauss-Seidel solver referred to as "QuickStep" or the pivoting Dantzig solver referred to as "WorldStep". The *threaded islands* parallelization strategy leverages the prevalence of multi-core CPU's in desktop computers and uses separate process threads to compute each island. This strategy will be most useful when simulation contains multiple robots that do not interact very often.

The second strategy attempts to speed up the constraint resolution algorithm within islands for the QuickStep solver. The ODE QuickStep solver is the default solver in Gazebo and solves constraints posed as a Linear Complementarity Problem (LCP). As an iterative, fixed time step projective solver, it is prone to position errors, such as interpenetration of objects. In order to correct these errors, an impulse is computed that is applied to the interpenetrating objects to push them apart. This method of position correction adds artificial energy into the system. To correct for this additional energy, two equations are solved. An error correcting LCP is used to correct the object position, while the velocity is updated without interpenetration error correction. These two equations are decoupled and can be solved in parallel, using two threads simultaneously. This comprises the second parallelization strategy. Further details of this method are given in the Appendix.

In future work, we may consider parallelization of collision detection, which is another potential performance bottleneck. It is not the focus of this work.

4 Results

This section presents results of the speed-up achievable by parallelization of physics using *threaded islands* and parallel position error correction. Each strategy is tested using some example simulated worlds with various numbers of models and levels of model complexity.

4.1 Test Scenarios

The effectiveness of the parallelization strategies described in this document depends on the scenario that is being simulated. For example, the *threaded islands* strategy can parallelize simulation of multiple non-interacting robots, while the position error correction strategy can parallelize simulation of individual robot. The following test scenarios vary the number and complexity of simulated entities to show the performance of each parallelization strategy.

Multiple Low Complexity Models

The *revolute_joint_test.world* file is used in Gazebo's automated test system. The world file includes eight instances of the *double_pendulum_with_base* model arrayed in a circle, as illustrated in Figure 1. The model consists of a base in contact with the ground that is connected to two links by revolute joints. The models are arranged in close proximity but do not contact each other. This scenario includes contact, articulation constraints, and multiple "islands".

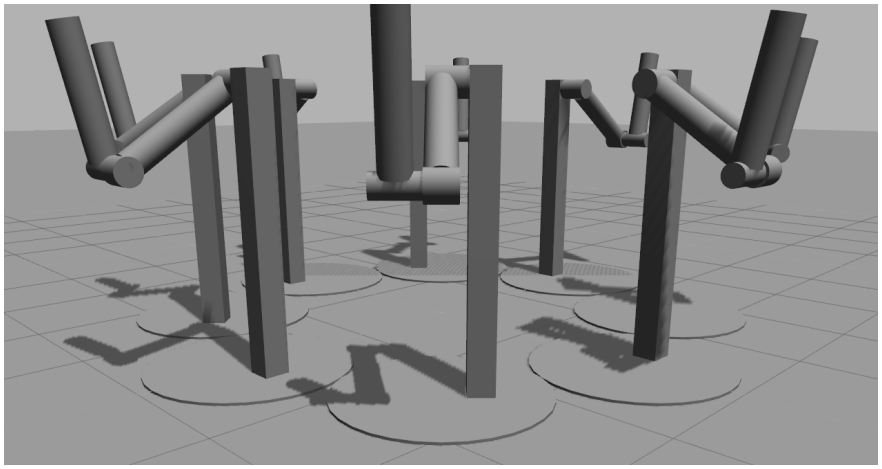

Figure 1: revolute_joint_test: a circle of eight double pendulums.

Single Complex Model

The *pr2.world* file includes a PR2 robot on a flat ground plane, as illustrated in Figure 2. There are no other objects with which to interact. The PR2 is a complex robot with 48 rigid bodies and 58 articulation joints. This scenario includes contact and articulation constraints but with only one "island".

Multiple Complex Models

The *dual_pr2.world* file includes two PR2 robots [1] on a flat ground plane, as illustrated in Figure 3. The robots do not interact with each other. This scenario is similar to *pr2.world* , but it includes two islands, so that the effect of each parallelization strategy can be compared with complex robots.

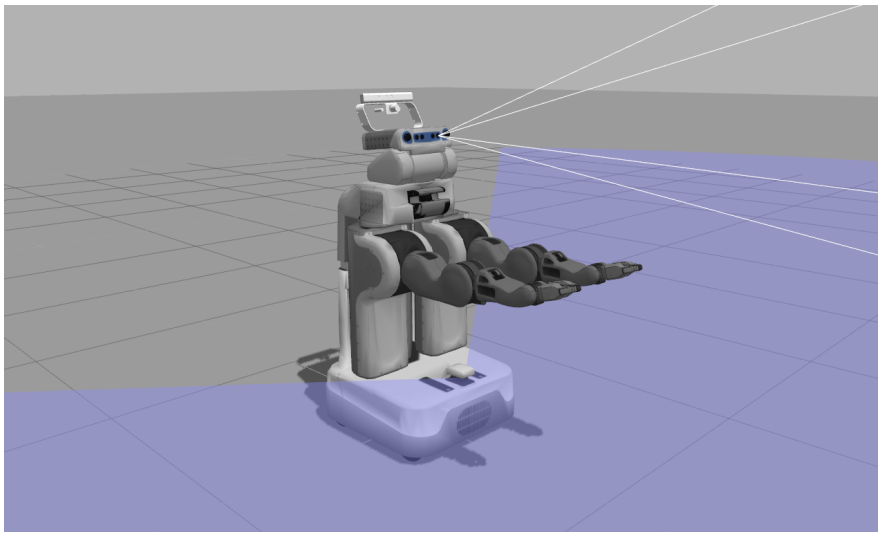

Figure 2: PR2: a solitary PR2 robot.

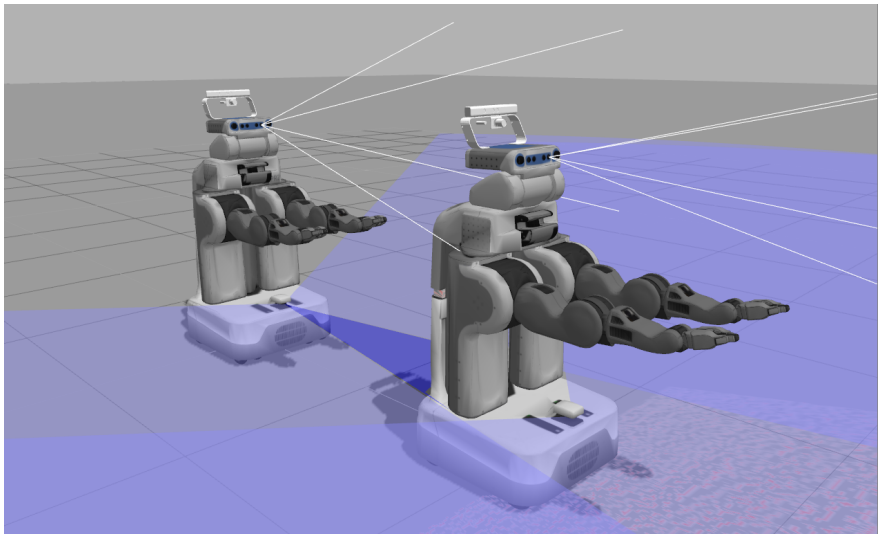

Figure 3: PR2:a pair of PR2 robots.

Many Single Rigid Body Models

The *drop_test.world* file includes a number of spherical bodies that fall due to gravity and purposely form many contacts between the bodies and the ground plane, as illustrated in Figure 4. The spheres are generated with random initial positions that do not initially interact, as shown in the left snapshot of Figure 4. The picture in the middle shows when the spheres fall onto the ground and generate contacts. The rightmost picture is after bodies sliding on the ground and they slide in different directions due to the initial random generation. Then they slide apart. This world file provides the scaling analysis of the parallelization strategies of the number of bodies and contacts in the environment.

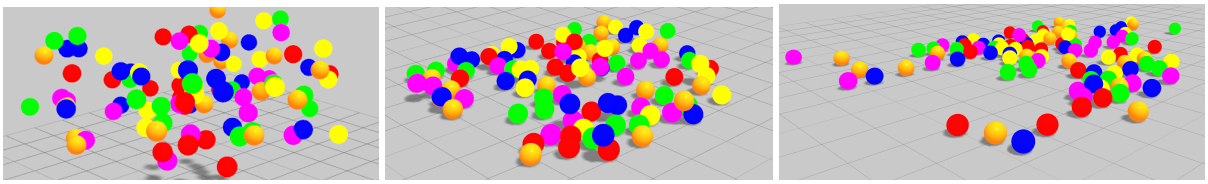

Figure 4: Drop test of 100 bodies to purposely form frictional contacts: left most picture shows simulation at the beginning (0 second), middle snapshot shows simulation at 1.5 seconds, right most picture is the simulation snapshot at 10.0 seconds. The simulation was run for 10 seconds.

Single High Complexity Model

The *joint_tree.world* file includes multiple joint trees, which is a hierarchical model of rigid bodies connected by ball joints. Each rigid body has two children connected by ball joints [2]. Figure 5 illustrates the case with five joint trees, each containing five layers of links and 31 ball joints in the system. As the number of joint trees changes, the number of islands in the system increases, while increasing the number of layers in each tree increases the complexity of each model.

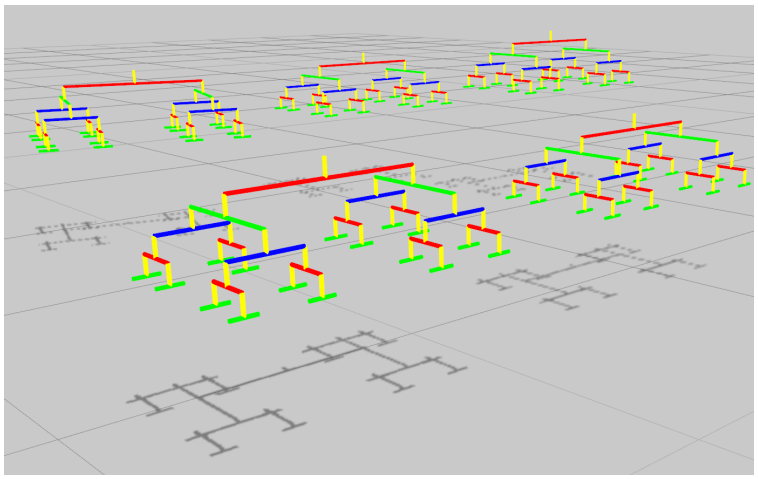

Figure 5: Joint tree with ball joints to connect links.

4.2 Performance measurement procedure

Gazebo is currently instrumented with high-resolution diagnostic timers at several parts of the inner loop. The timer resolution depends on the hardware in use and is approximately 100ns on our test machines. To prevent performance degradation during normal usage, the timers are disabled unless the "ENABLE_DIAGNOSTICS" symbol is defined during compilation. For each simulation step, the elapsed time is measured and used to compute the following statistics incrementally: mean, minimum, maximum, and variance of each diagnostic timer. The statistics are computed using the "math::SignalStats" class.

The instrumented simulations are conducted on quad-core desktop computers without real-time guarantees. As such, the computational time can vary between tests for a variety of reasons. To account for this variation, simulations are repeated multiple times for each set of parameters to provide additional statistics. The "run_diagnostic.bash" and "parallel_tests.bash" scripts in the "diagnostics_scpeters" branch were used to run simulations with 1 ms step size and 10 second duration. Simulations were repeated 10 times for each set of threading parameters.

In the following sections, results are presented based on the time elapsed during the "ODEPhysics::UpdatePhysics" diagnostic timer. This timer corresponds to the physics portion of the inner loop. The average (mean) and best-case (minimum) step times are presented in plots. Box plots are used to represent the distribution of the average and best-case step times across the 10 simulation trials. The speed-up plots use the median value from the box plot.

4.3 Parallelization results: revolute_joint_test.world

The effects of "threaded islands" were measured for the revolute_joint_test.world with 0 threads (control) as well as $1-6$ threads. The distribution of average and best-case times for "ODEPhysics::UpdatePhysics" are shown in Figure 6. The median speed-up is shown as the solid lines in Figure 7. The clear trend in these two plots is that using 1 island thread reduces performance by 20-30% relative to the control case of 0 island threads, while 2 or more threads increase average performance 50% or more. The best-case performance can be improved by up to 90%, which represents the potential for further performance improvement in the threading implementation.

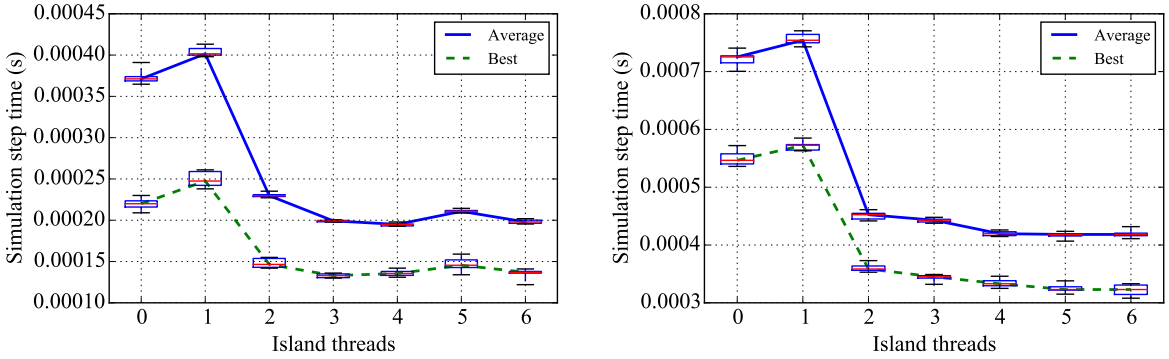

Figure 6: Average and best-case simulation step time in revolute_joint_test.world without (left) and with (right) threaded position error correction enabled. Both tests are run with island threading enabled.

Results for threaded position error correction for revolute_joint_test.world.

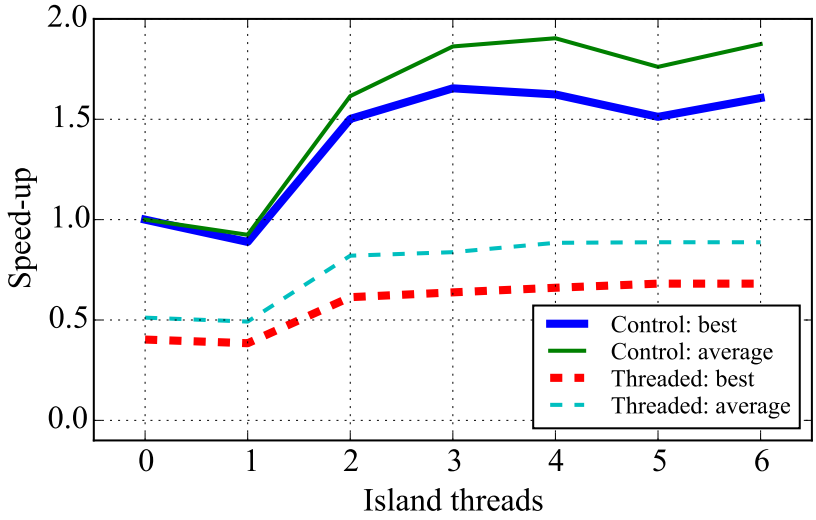

Figure 7: Median of average and best-case speedup for 10 trials of revolute_joint_test.world with (Threaded) and without (Control) threaded position error correction.

4.4 Parallelization results: pr2.world

The pr2.world is run with 0 island threads (control), as well as $1-4$ threads. Additionally, the effects of threaded position error correction were measured for pr2.world. The average and best simulation step time is shown in figure 8. For the single PR2 test, the simulation step time is higher with a non-zero number of island threads, with or without threaded position error correction enabled. Threaded islands don't help the single PR2 scenario since the complex PR2 model cannot be partitioned and solved simultaneously over several threads. Therefore, it wastes more time to allocate and clean up for both island and position threads. Figure 9 shows that enabling threaded position error correction leads to a 15% speed-up.

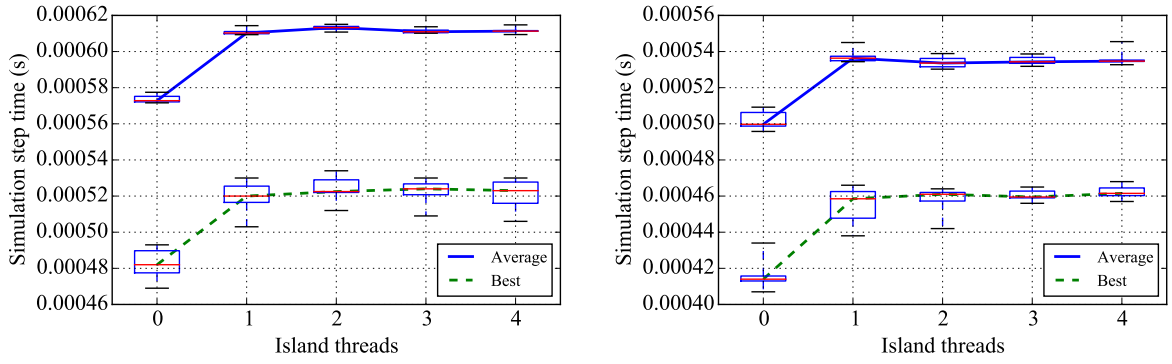

Figure 8: Average and best-case simulation step time in pr2.world without (left) and with (right) threaded position error correction enabled. Both tests are run with island threading enabled.

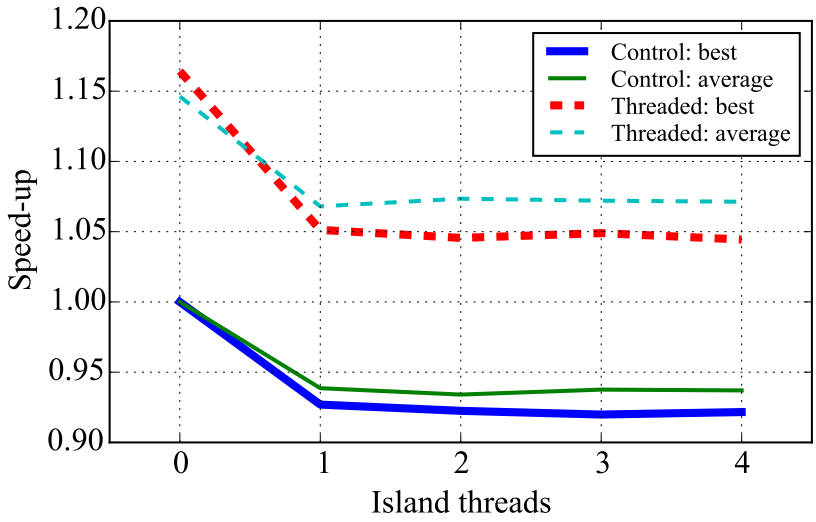

Figure 9: Median of average and best-case speedup for 10 trials of pr2.world with (Threaded) and without (Control) threaded position error correction.

4.5 Parallelization results: dual_pr2.world

Simulation step times for the dual_pr2.world scenario are shown in Figure 10. Again, there is a penalty to specifying 1 island thread, but there is a substantial improvement when using 2 threads. As island thread number increases from 2 to 4, there is no speed up. Since we have only 2 PR2 robot, 2 island threads will be the optimal case, each solving one PR2 model. As we keep increasing the thread number from 2, the model cannot be split up any more, so there is no speed up.

Figure 11 shows the speed up factor with and without threaded position error correction. As with the single pr2 scenario with no threaded islands, there is about a 15% speed-up when using threaded position error correction. With 2 thread islands, the position thread enabled best case has a speed up factor around 1.97, while the case without is around 1.83. When considering the average simulation step time, however, the case without threaded position error correction (control) is faster, with a speed up factor of 1.82 compared to 1.59 with threaded position error correction. Note that using 2 island threads and 2 threads for position error correction will dedicate 4 threads just for physics, which may cause contention problems in a quad-core processor, since Gazebo has other threads as well, in addition to the other system processes. This may explain the reduction in average speed.

This results strongly verify the analysis for single PR2 robot scenario, the optimal number of island threads to use for fastest parallel physics performance is strongly related to the number of individual models that don't have constraints with each other.

4.6 Parallelization results: drop_test.world

Figure 12 shows the average simulation step time for dropping test with different number of thread islands. To rule out randomness of the results, we duplicate run the same simulation for ten times. Each plot in Figure 12 corresponds to different number of bodies, labelled in the top right legend box. When thread island is 0, it represents the situation

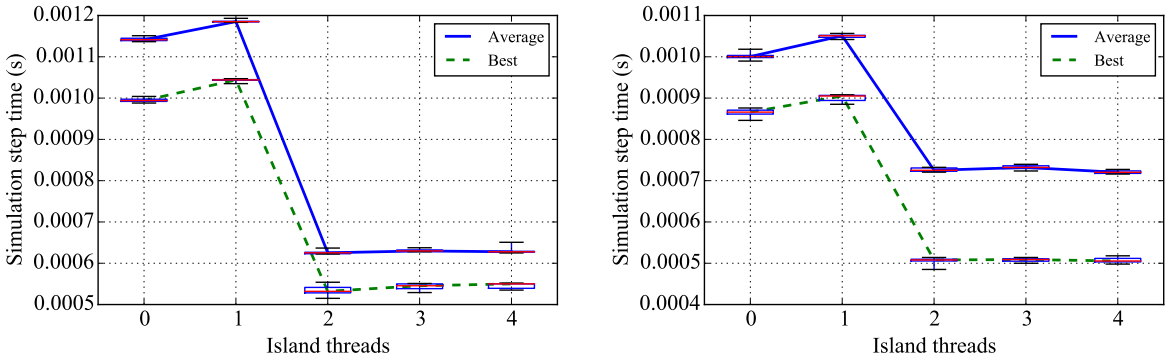

Figure 10: Average and best-case simulation step time in dual_pr2.world without (left) and with (right) threaded position error correction enabled, both tests are run with island threading enabled.

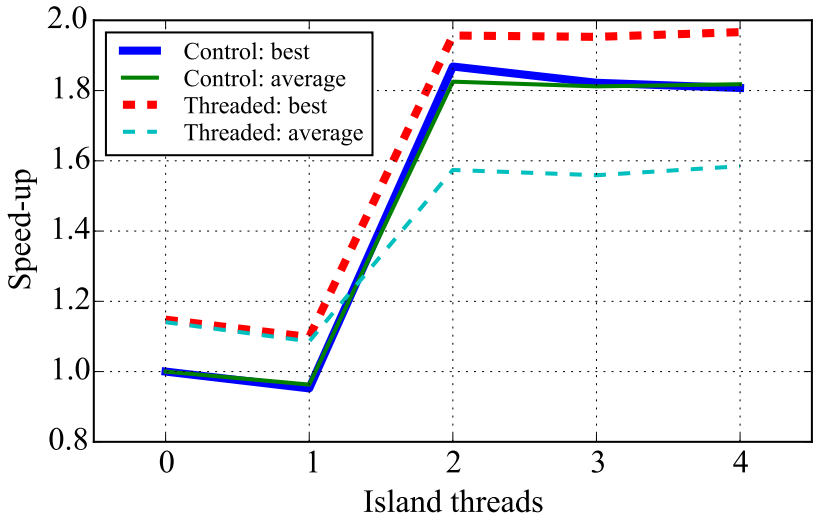

Figure 11: Median of average and best-case speedup for 10 trials of dual_pr2.world with (Threaded) and without (Control) threaded position error correction.

where we don't use the parallelization strategies. From figure 12, the time it takes to run the simulation has an increase of $20\% - 40\%$ from 0 to 1 thread, which is due to the overhead of passing data to a single thread instead of using the main thread. With one island thread, the time spent on communication outweighs the parallelization. With thread number increasing, the task is distributed onto more threads and each thread get a smaller part of the task, compared with when thread number is smaller. Then the time saved by parallelization is greater than the communication time, therefore, we have a net decrease in the simulation time with more number of threads.

Dropping test speed up results in Figure 13 actually makes it harder to parallel due to the coupling between all those contacts, which makes the case with many bodies contact problem as one whole simulation scenario that is hard to split into multiple thread islands.

We take 100 bodies as an example, and show the results of all 10 experiments with a box plot. The result is showed in Figure 14. We could see the total simulation time is 0.0065 seconds without parallelization, while when we use five threads, it decreased to around 0.003 second. The performance has an increase around 54%.

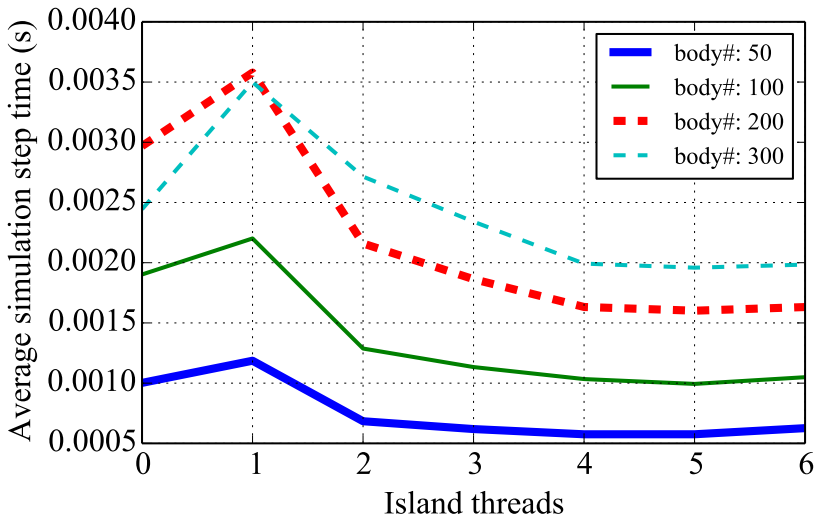

Figure 12: Average simulation step time for dropping test.

4.7 Parallelization results: joint_tree.world

Figure 15 and Figure 16 show the average and minimum simulation step time for the joint tree scenario, respectively. Besides the average simulation step time, we also recorded the minimum simulation step time, since the system size doesn't change once the simulation starts. In figure 15, each plot represents different scenario, where the joint tree number represents the individual model that doesn't have interaction with each other, and the layer number stands for the complexity of each model. Suppose the layer number is n, then we will have 2^{n-1} ball joints in each individual model.

Figure 16 depicts the minimum simulation step time for the various joint tree systems, which represents the best-case of simulation performance.

As for speed up factor, the configuration with 7 joint trees, each having 4 layers has a maximum speed up factor of 2.3, using 6 threads, while with the best case of simulation step time, the speed up factor is 3 with 7 threads. All the

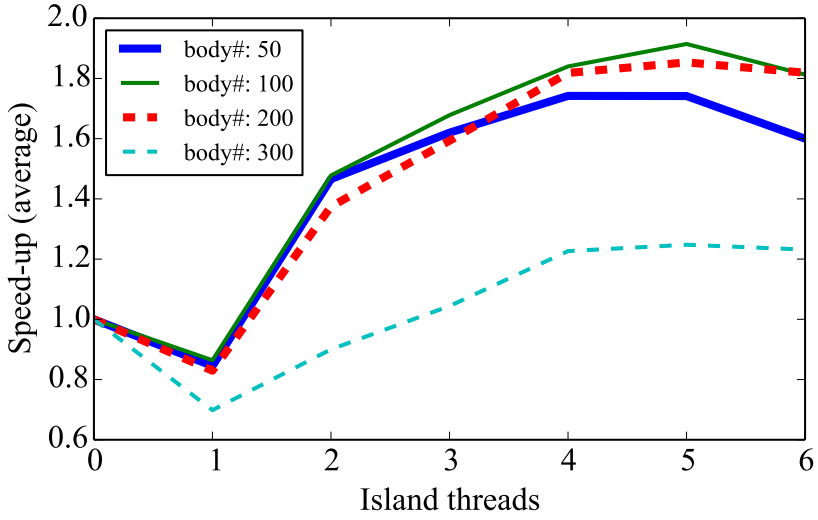

Figure 13: Speed up factor with the average simulation step time versus the number of thread islands used.

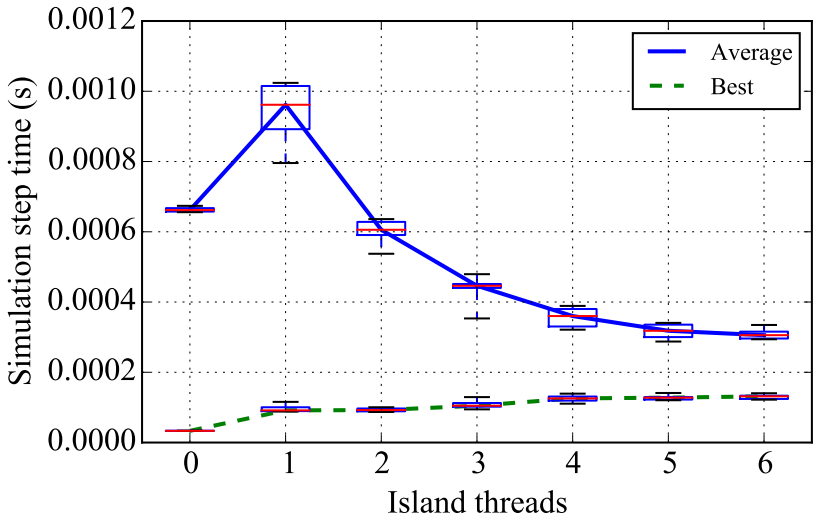

Figure 14: Average and best-case simulation step time in drop_test.world with 100 bodies.

other configurations have a speed up factor around 2.

The speed up factor for joint tree test tends to be larger than the drop tests, which is due to the smaller number of frictional contacts, which is one of the bottlenecks in improvement of speed of solution algorithms in ODE, especially with friction.

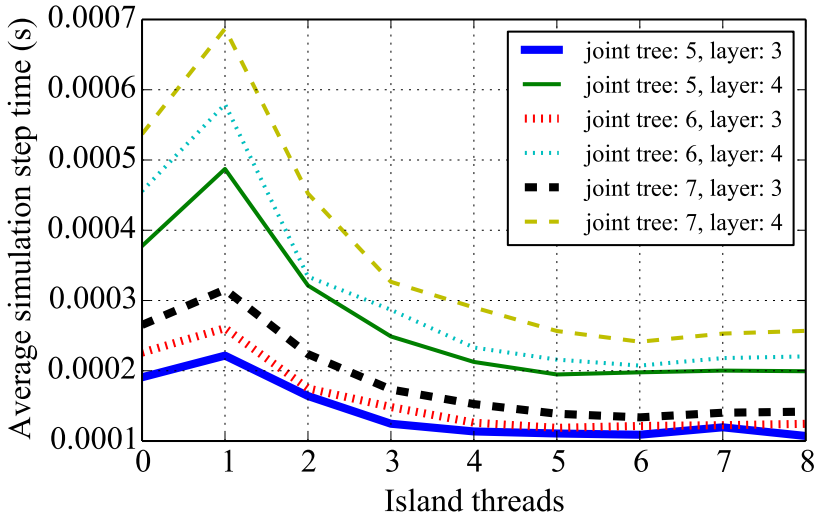

Figure 15: Average simulation step time for joint tree test.

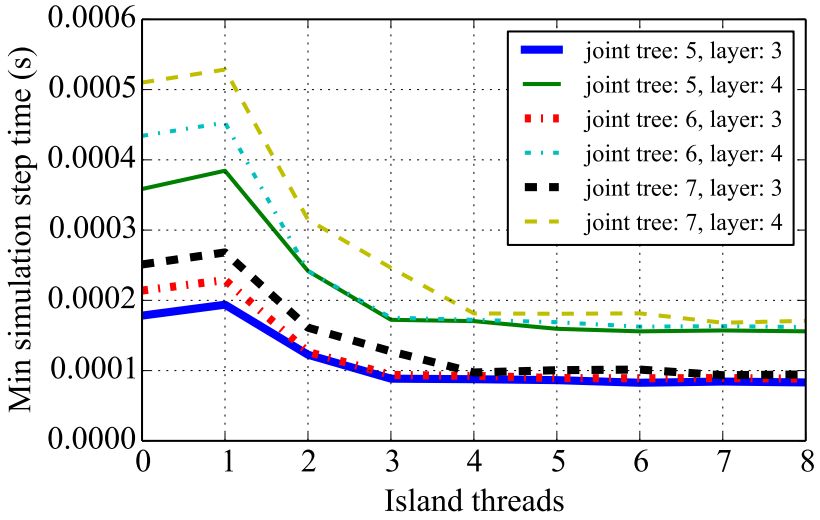

Figure 16: Minimum simulation step time for joint tree test.

Figure 19 shows the simulation step time for a specific test with 7 joint trees and each has 4 layers. The red line in the boxed plot shows the median of 10 experiments to rule out randomness. The lowest and highest bar bounds the simulation step time limits. The boxes cuts at 25% and 75% of the 10 simulation. We could see a constant decrease in average simulation time from 1 to 7 threads increase.

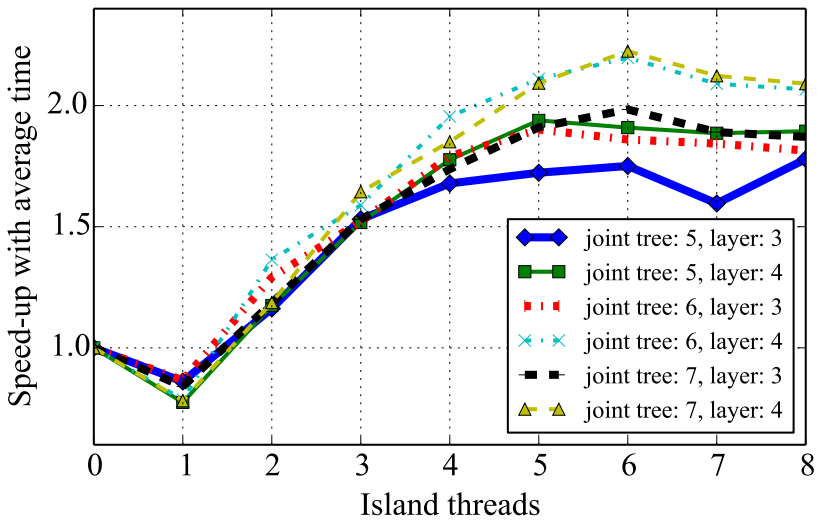

Figure 17: Speed up factor for joint tree test (Average simulation step time).

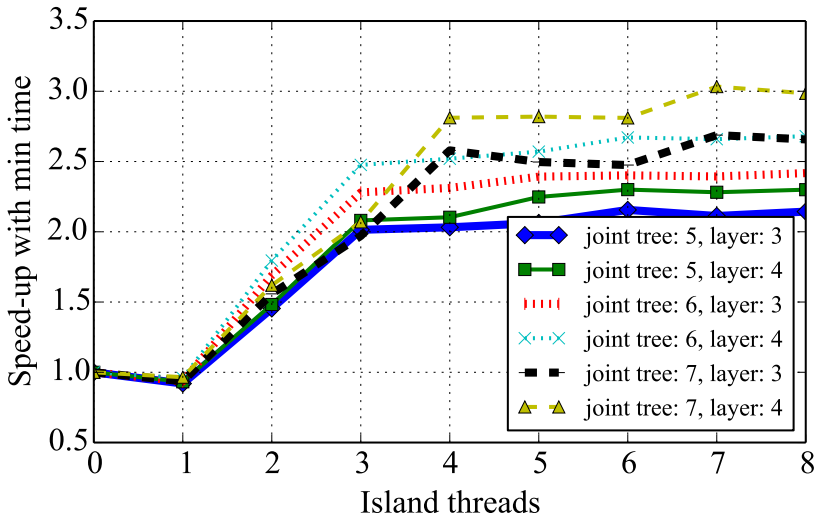

Figure 18: Speed up factor for joint tree test (minimum simulation step time).

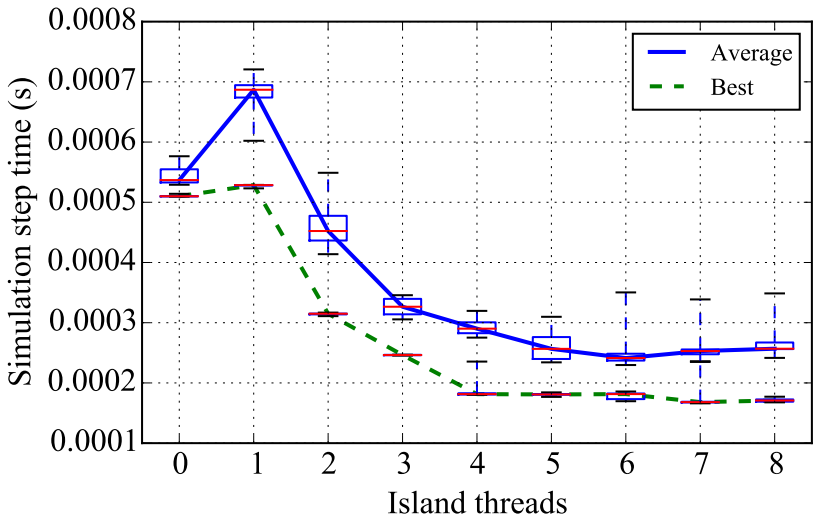

Figure 19: Average and best-case simulation step time in joint_tree.world with 7 joint trees and each has 4 layers.

References

[1] W. Garage, "Personal robot 2 (pr2)." http://www.willowgarage.com.

[2] J. Bender and A. A. Schmitt, "Fast dynamic simulation of multi-body systems using impulses," in *In Virtual Reality Interactions and Physical Simulations (VRIPhys)*, pp. 81–90, 2006.

[3] M. a. Sherman, A. Seth, and S. L. Delp, "Simbody: multibody dynamics for biomedical research," *Procedia IUTAM*, vol. 2, pp. 241–261, Jan. 2011.

[4] J. Baumgarte, "Stabilization of constraints and integrals of motion in dynamical systems," *Computer Methods in Applied Mechanics and Engineering*, vol. 1, pp. 1–16, June 1972.

5 Appendix: parallel position error correction

Rigid body dynamics solvers with fixed time stepping schemes will inevitably encounter instances where two rigid bodies intersect. When that happens, one approach is to backup simulation in time and take smaller time steps until a non-penetrating contact has been made. Physics engines such as Simbody [3] uses variable time stepping approach such as this to address the issue of resolving collision interpenetration.

Constraint violation can also be caused by drift in position from numerical errors during integration and residual errors in unconverged iterative LCP procedures. The effect of drift may be less obvious with pivoting methods than iterative methods depending on the convergence of the iterative LCP solver such as PGS. Nevertheless, the constrained position will most likely drift if no correction is applied.

To improve constraint satisfaction in the presence of these error sources, ODE adds a position constraint correction term. The position constraint error is evaluated for each constraint and expressed at timestep n as $\bar{h}^n$. It is added to the velocity constraint equation with coefficient **ERP** (also known as error reduction parameter **ERP** inside ODE).

$$\bar{\bar{J}}\bar{v} + \frac{\mathbf{ERP}}{\Delta t}\bar{h}^n = \bar{c} \tag{1}$$

This term can be considered a form of Baumgarte stabilization [4]. It is used to restore constraint error to zero in one time step when $\mathbf{ERP} = 1$ with a first-order Euler integrator, though values less than 1 are used in practice.

The constraint error correction term written as part of the main governing equations of motion is shown below:

$$[\bar{\bar{J}}\bar{\bar{M}}^{-1}\bar{\bar{J}}^T]\bar{\lambda}^{n+1} = \frac{\mathbf{ERP}\bar{h}_{corr}}{\Delta t} - \bar{\bar{J}}[\frac{\bar{v}^n}{\Delta t} + \bar{\bar{M}}^{-1}\bar{f}_{ext}]. \tag{2}$$

The problem with the existing correction method in ODE is that the **ERP** term adds non-physical energy to the system every time an interpenetration occurs. Alternatively, if **ERP** is zero, the solution will not correct for interpenetration

and the rigid bodies in contact may *drift* into deeper constraint violations. A method similar to the Split Impulse method was introduced in ODE to cope with position errors caused by fixed time step interpenetration, unconverged iterative PGS residual and numerical integration errors.

For this method, two LCP equations are solved side by side, with **ERP** to yield $\bar{v}^{n+1}_{\textbf{ERP}}$ and without **ERP** to yield $\bar{v}^{n+1}$.

Below is an intermediate step that solves for accelerations with and without correction terms:

$$\bar{\bar{J}}\bar{a_c}^{n+1}_{\textbf{ERP}} = \frac{\textbf{ERP} \cdot \bar{h}}{\Delta t^2} - \bar{\bar{J}}\bar{\bar{M}}^{-1}\bar{f}^{n+1}_{ext} \tag{3}$$

$$\bar{\bar{J}}\bar{a_c}^{n+1}_{\textbf{ERP=0}} = -\bar{\bar{J}}\bar{\bar{M}}^{-1}\bar{f}^{n+1}_{ext} \tag{4}$$

to obtain accelerations, $\bar{a_c}_{\textbf{ERP}}$ and $\bar{a_c}_{\textbf{ERP=0}}$.

The two equations are solved in parallel for efficiency. Once velocity solutions are obtained, the solution without correction terms (**ERP**) is used to update velocity

$$\bar{p}^{n+1} = \bar{p}^{n} + \Delta t \left(\bar{a_c}^{n+1}_{\textbf{ERP=0}}\right) \tag{5}$$

while the solution with correction terms **ERP** enabled is used to update position:

$$\bar{p}^{n+1} = \bar{p}^{n} + \Delta t \left(\bar{a_c}^{n+1}_{\textbf{ERP}}\right) \tag{6}$$

.

The proposed position projection correction approach effectively *teleports* the overlapping objects away from each other without introducing excess kinetic energy into the system. At the velocity level, contact constraints appear as non-penetrating inelastic impacts if PGS converges fully.

For example, a box on the ground plane with initial collision interpenetration of 1 centimeter will correct its position until resting contact is achieved without gaining energy if position projection is turned on. Figure 20 shows box position trajectory and velocity with and without position projection correction.

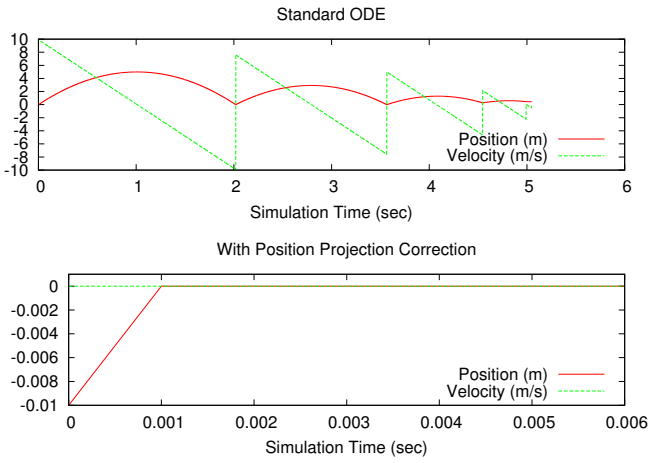

Figure 20: Position and velocity of box model on the ground plane with initial interpenetration of 1 meter. Plot shows trajectory of the box with and without position projection correction. For standard ODE runs, **ERP** is 1.

In addition to more stable interpenetration correction, position projection correction is ideal for modeling completely inelastic impacts as demonstrated in figure 21.

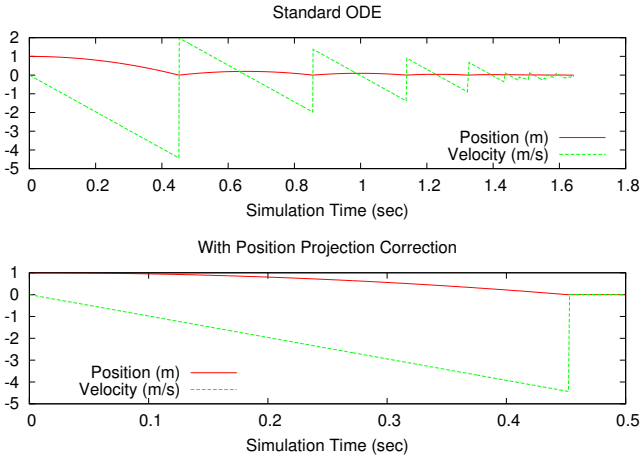

Figure 21: Position and velocity of a box model impacting ground plane with and without projected position correction. For standard ODE runs, **ERP** is 1.