Jolt Physics Multicore Scaling

Jorrit Rouwé – 22 Jan 2022 (updated Mar 10th 2023)

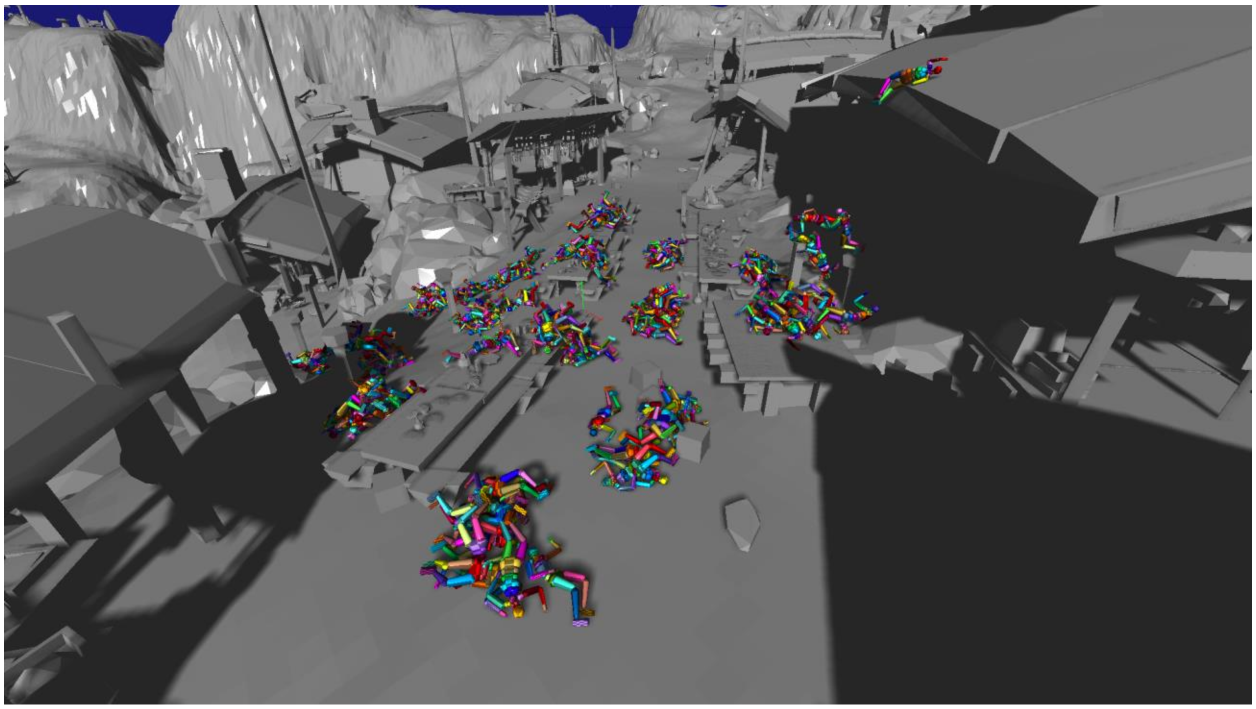
Introduction

This article will investigate how Jolt Physics performs with different CPU architectures and varying amount of CPU cores. It will also compare Jolt Physics with two other popular physics engines: PhysX and Bullet.

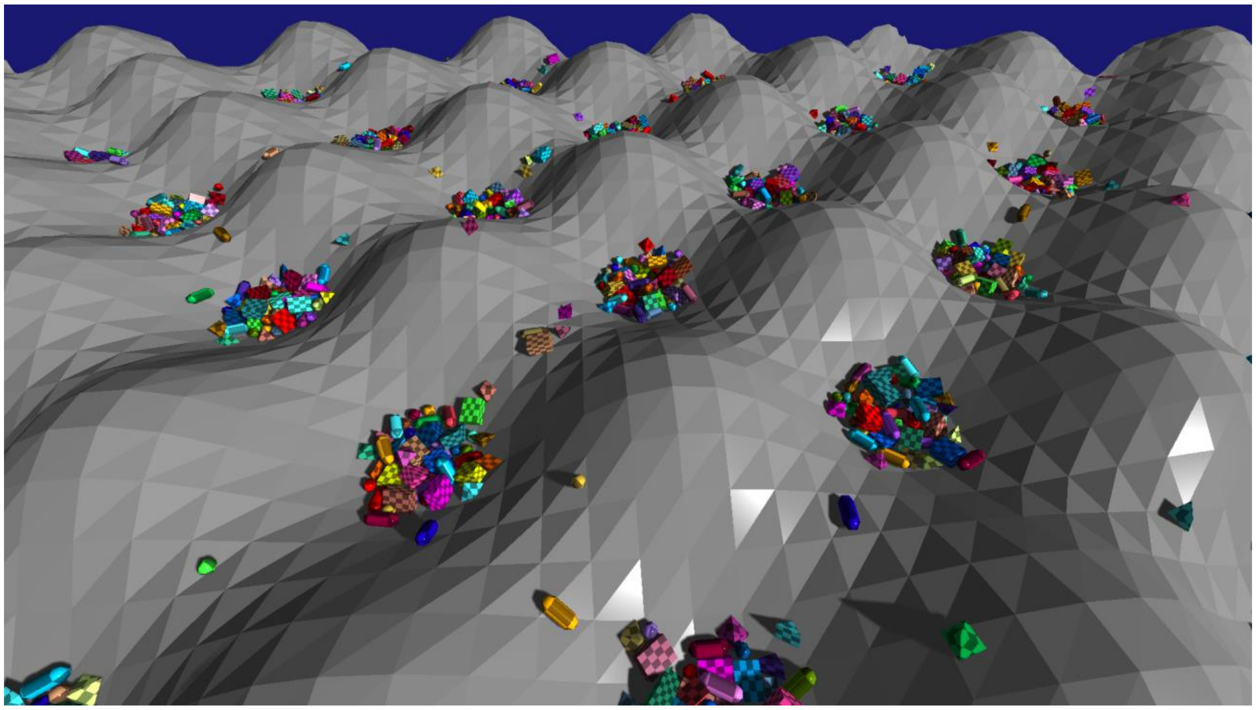
Test Setup

The graphs in this document were produced by PerformanceTest, this test has 2 different scenes:

1. A scene with 16 piles of 10 ragdolls (3680 bodies) with motors active dropping on a level section.



2. A simpler scene of 484 convex shapes (sphere, box, convex hull, capsule) falling on a 2000 triangle mesh.



The first scene gives a better indication of real-world performance, the second scene is simpler and easier to port to other physics engines. An implementation of this second scene has been made for PhysX 4.1 and Bullet 3.21 (click to go to the source code).

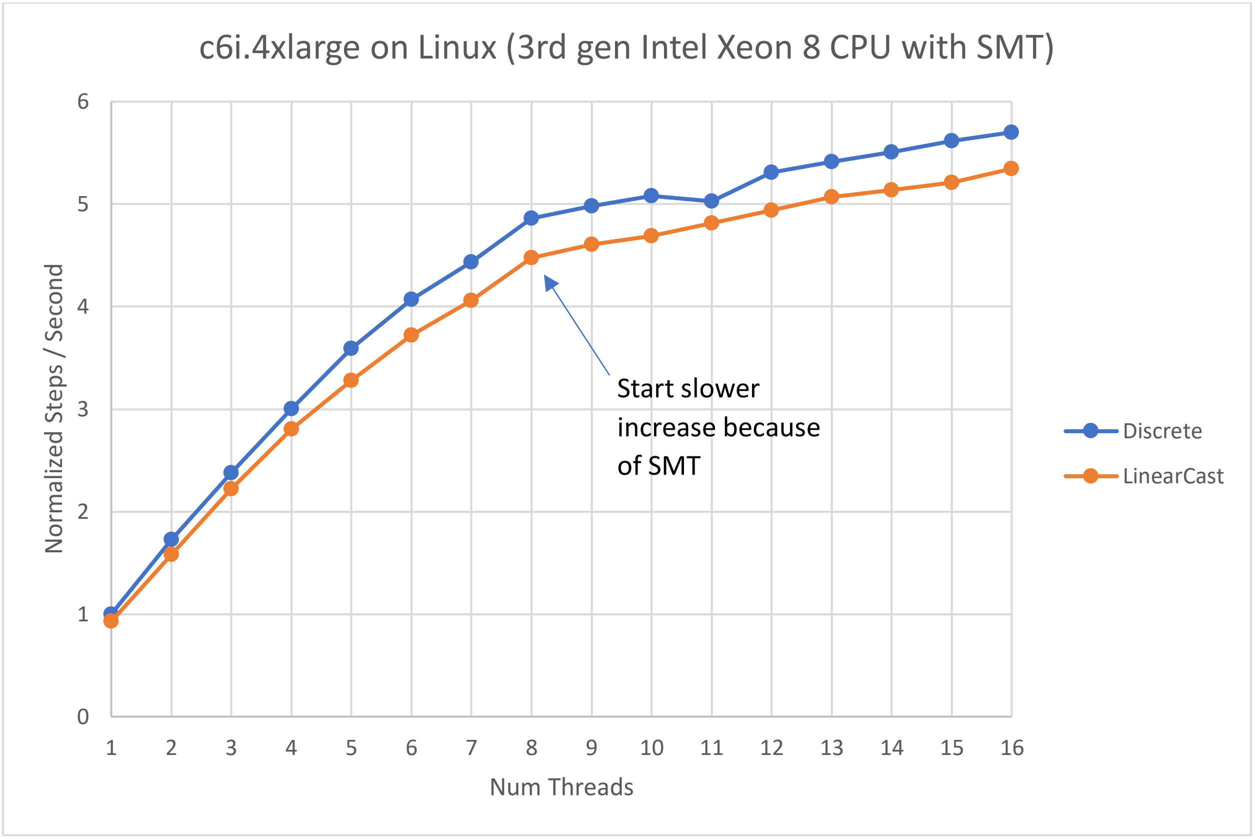
All tests show the amount of simulation steps per second versus the number of threads with which the simulation ran. Each test ran multiple times, the best result was kept minimizing the influence of other processes. The tests were first done with Continuous Collision Detection (CCD) turned off and then repeated with CCD on. Note that CCD off is called 'Discrete' in Jolt, CCD on is called 'LinearCast'.

The tests that ran on Amazon EC2 (AWS) were done on Compute Optimized instances (c-series) where possible. The General Purpose (m-series) were found to be less suitable for performance measurements as the results fluctuated a lot. Unfortunately, the latest generation AMD processors only come in a General Purpose flavor (m6a) so that's the only exception. The EC2 instances were running Ubuntu 20.04 and the code was compiled with Clang 12 on the 'Distribution' version.

The tests that ran on an i7-7700HK were compiled with MSVC 2022 version 17.0.5. PhysX and Bullet were both compiled in their 'Release' mode.

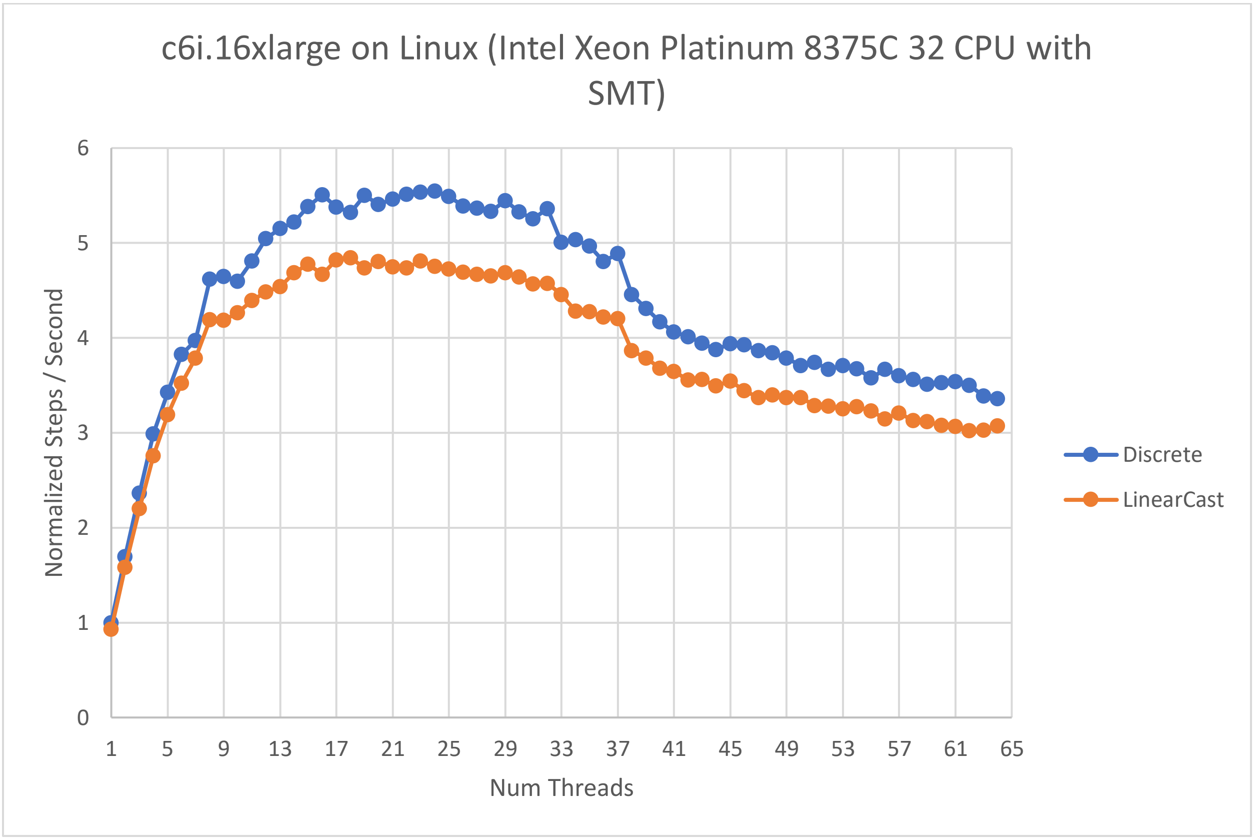
Scaling of the Ragdoll Scene

First, we will show how Jolt Physics scales with multiple threads on EC2 instances.



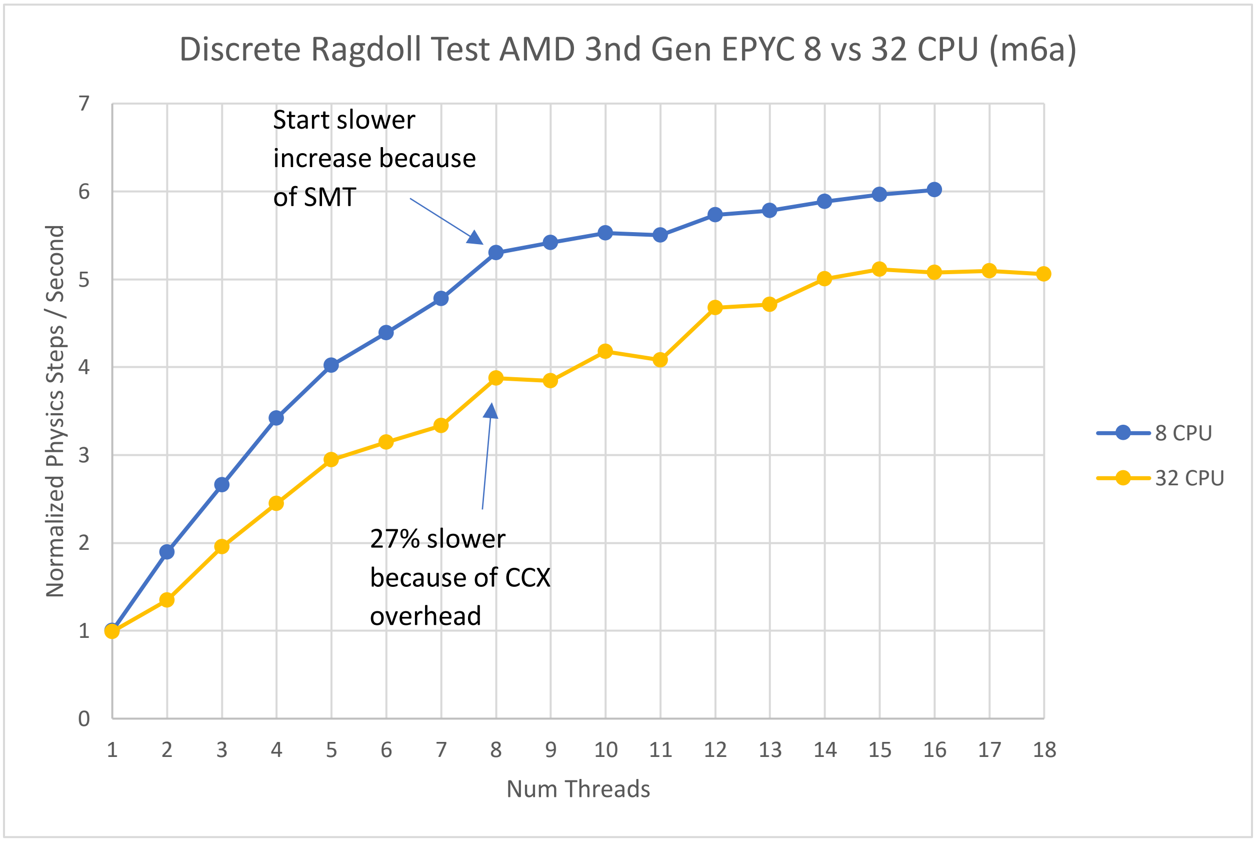
The graph above shows the amount of physics simulation steps per second normalized to 1 (1 core without CCD = 1). You can see that Jolt is 4.9x faster with 8 threads than it was with 1 thread. Using Simultaneous Multithreading (SMT) increases this to a 5.7x speedup with 16 threads. The system scales equally well with and without CCD although CCD (LinearCast) is obviously a bit slower.

When going to extreme amounts of CPU cores we can see that Jolt does not scale infinitely.



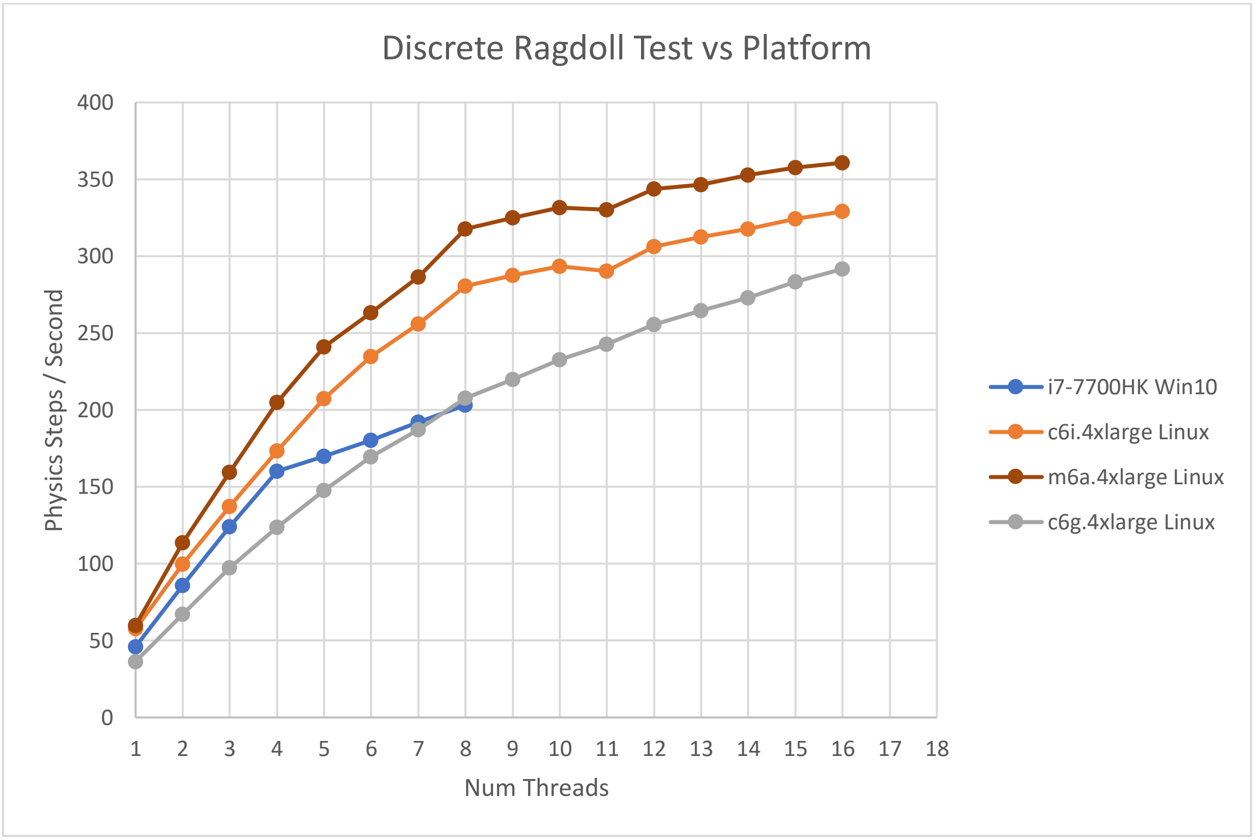
After about 16 CPU cores Jolt stops scaling and eventually deteriorates. At this point the memory bus becomes a big bottleneck and the lock free operations that are used to manage the contact cache dominate the simulation update.

On an AMD platform there is another thing to be aware of.



AMD processors consist of multiple Core Complexes (CCX). Memory operations (and especially atomic operations) cost a lot more when they cross CCX borders. Currently Jolt is not CCX aware. As you can see at 8 threads the simulation on 32 CPUs (4 CCX) is 27% slower than the same machine with 8 CPUs (1 CCX) just because jobs were scheduled at random across CCXs. To avoid this overhead, set the thread affinity of the job scheduler to use only a limited set of CCXs.

An overview of the absolute performance of the ragdoll scene for all tested platforms.



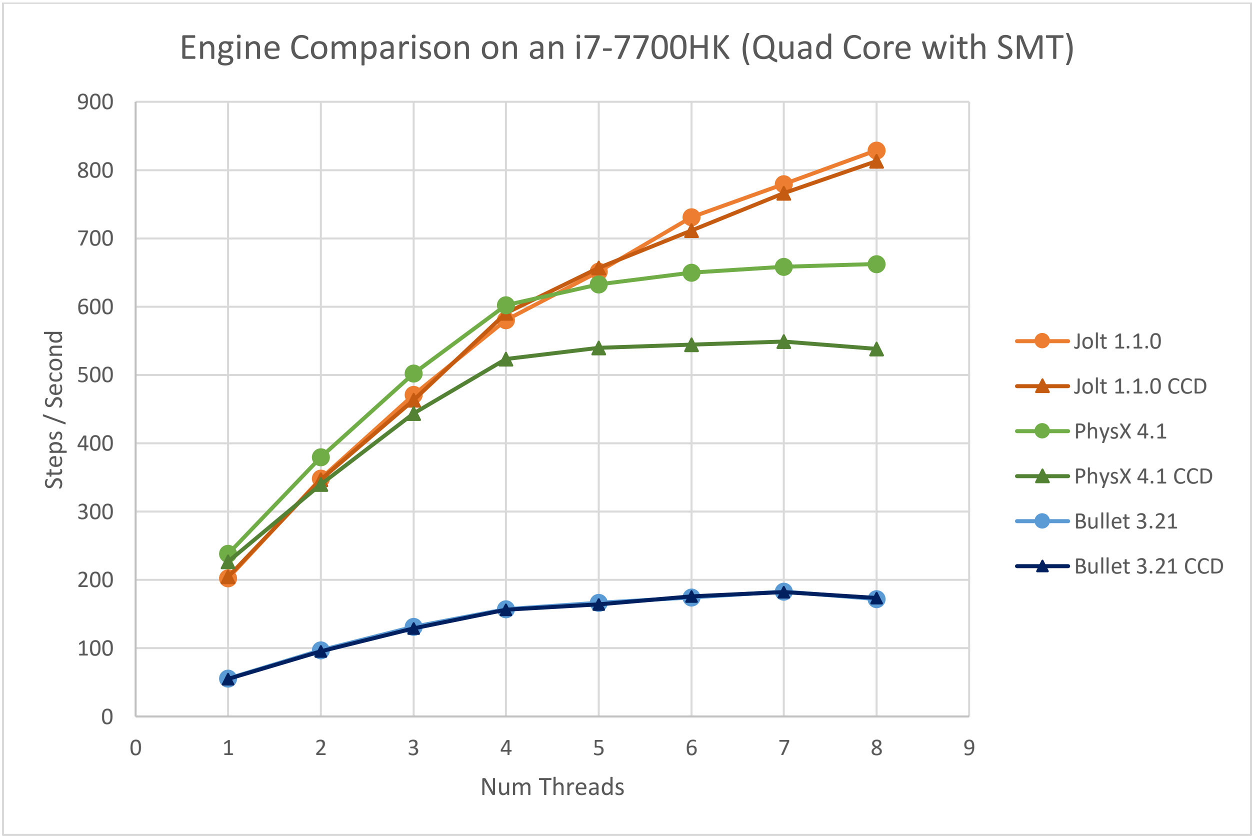
This includes a Windows 10 laptop with an i7-7700HK, Intel Xeon 3rd Gen (c6i), EPYC 3rd Gen (m6a) and AWS ARM Graviton 2 (c6g, note no SMT) processors. If Jolt runs on a single CCX the 3rd Gen AMD EPYC wins.

Comparison Between Physics Engines using the Convex Vs Mesh Test

As stated, a separate test was implemented for 2 other physics engines. To keep the comparison fair, we have tried to configure them in a similar way:

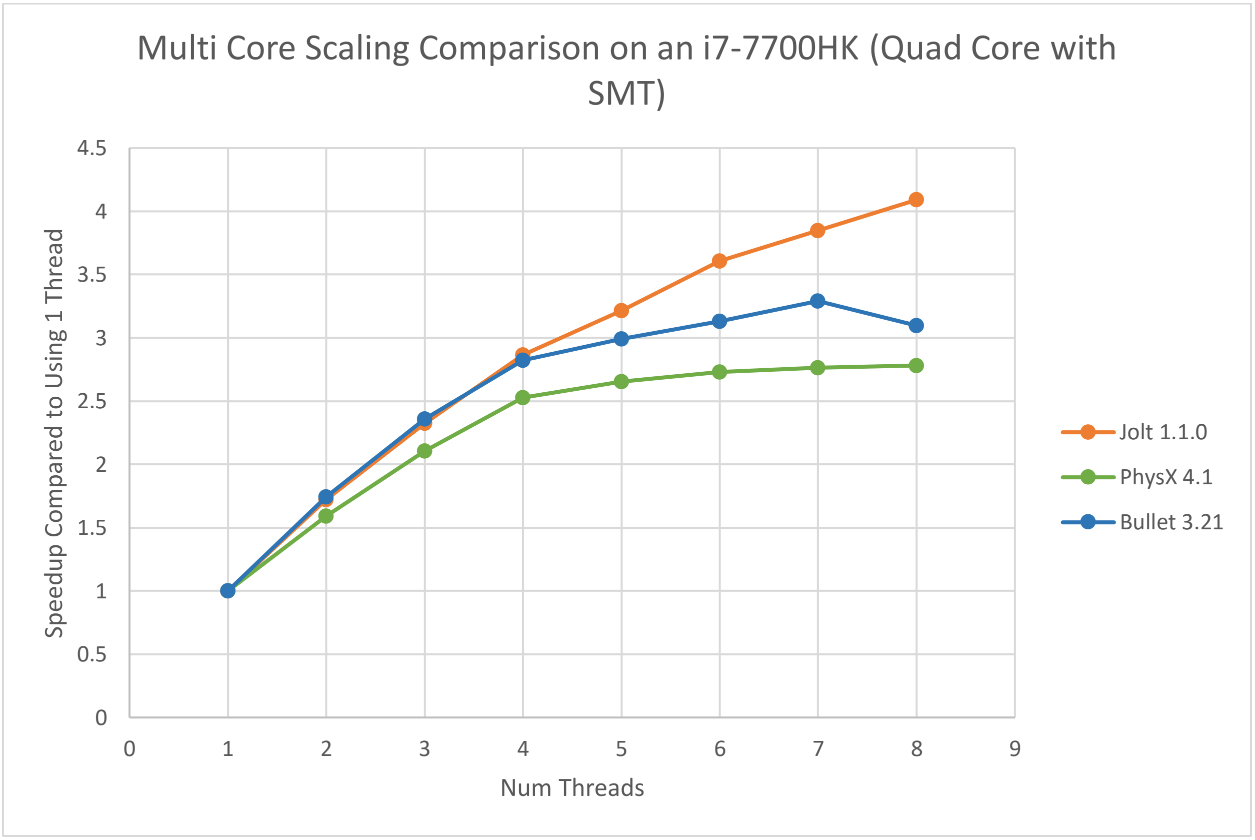
- PhysX was configured to use only 1 CCD pass (ccdMaxPasses) since both Bullet and Jolt support only 1 pass. Note that Bullet doesn't use the actual shape for CCD collision detection but simplifies the shape to a sphere. Jolt and PhysX seem to be comparable in terms of how CCD is implemented: Using the actual shape to do a linear cast and stopping at the first collision.
- PhysX was kept at the default solver iterations count (4 position + 1 velocity), Bullet was set to 5 solver iterations (it applies position correction every iteration) and Jolt was configured to 4 velocity + 1 position iteration (the position iterations are much more expensive for Jolt, for PhysX the velocity iterations seem to be more expensive).

This is the result of the test:



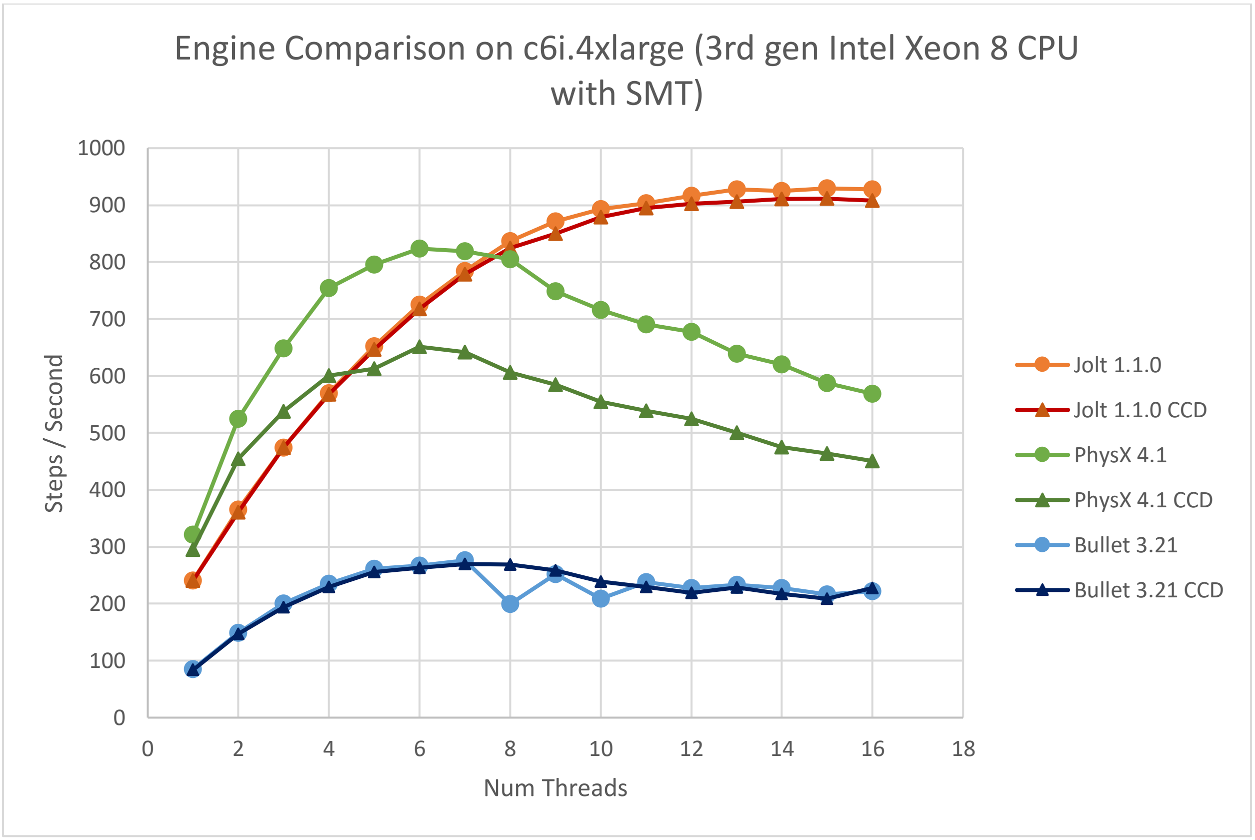
The first thing to note is that PhysX has a much higher penalty for CCD at higher thread counts than Jolt and Bullet. The test doesn't really stress the CCD system as it contains relatively large shapes moving at a speed that doesn't cause them to tunnel through the mesh terrain. Jolt and Bullet do some CCD tests, but the performance is impacted very little by it.

Calculating the relative speedup per thread:



We can see that, for this scene, Jolt scales better than the other physics engines (although its single threaded performance is worse than that of PhysX).

Repeating the test on an c6i.4xlarge with 8 CPUs shows that, for this scene, both PhysX and Bullet stop scaling at around 6-7 CPU cores:



Conclusion

We have shown that Jolt scales well with multiple cores and for a very simple test scene achieves similar performance as PhysX. Note that you should always compare physics engines on the actual scene that you want to run as each engine has its strengths and weaknesses.

Update Nov 13th 2022: PhysX limits the number of threads it uses based on the scene size, so it can scale up beyond what is shown in the graphs above, but only if the test scene is made larger. This reinforces that you should always profile your specific scenario. For more information see the following discussion.

Update Mar 10th 2023: Removed remark that Jolt does not scale well when all objects are on a single pile. This has been fixed.