Fast DEM collision checks on multicore nodes

Konstantinos Krestenitis, Tobias Weinzierl, and Tomasz Koziara *

School of Engineering and Computing Sciences
Durham University
Great Britain
{konstantinos.krestenitis,tobias.weinzierl}@durham.ac.uk

Abstract. Many particle simulations today rely on spherical or analytical particle shape descriptions. They find non-spherical, triangulated particle models computationally infeasible due to expensive collision detections. We propose a hybrid collision detection algorithm based upon an iterative solve of a minimisation problem that automatically falls back to a brute-force comparison-based algorithm variant if the problem is ill-posed. Such a hybrid can exploit the vector facilities of modern chips and it is well-prepared for the arising manycore era. Our approach pushes the boundary where non-analytical particle shapes and the aligning of more accurate first principle physics become manageable.

1 Introduction

Discrete Element Methods (DEM) are a popular technique to model granular flow, the break-up of brittle material, ice sheets, and many other phenomena. They describe the medium of interest as a set of rigid bodies that interact with each other through collisions and contact points. The expressiveness of such a simulation is determined on the one hand by the accuracy of the physical interaction model. On the other hand, it is determined by the accuracy of scale: the more rigid bodies (particles) can be simulated the more accurate the outcome.

Many DEM codes restrict themselves to analytical shape models: Their particles are described by some analytical function; most of the time spheres. Furthermore, they stick to explicit time integrators (cmp. [1] and references therein). Whenever particles are close to each other, i.e. their distance underruns a given threshold, they are assumed to be in contact. An interaction model then realises two types of physics. On the one hand, it mitigates the real-world impact of collision, friction, and so forth. On the other hand, it mitigates the fact that real particles are not spherical/analytical [6].

While the distributed memory parallelisation of DEM codes through classic domain decomposition is well understood and the codes scale (see [5] as an example), most codes refrain from modelling particles as irregularly shaped objects

* This work has been sponsored by EPSRC (Engineering and Physical Sciences Research Council) and EDF Energy as part of an ICASE studentship (award ref 1429338). It made use of the facilities of the Hamilton HPC Service of Durham University. All software is freely available from https://github.com/KonstantinosKr/delta (pronounced/written Δ).

and, thus, from eliminating the second role of the interaction model, as they already spend a majority of their compute time in collision detection. Iglberger et al. [5] report 31–34% within a multiphysics setting, while Li [7] for example reports even 90%. Detection becomes significantly more complicated once we switch from sphere-to-sphere or ellipsoid-to-ellipsoid checks to the comparison of billions of triangles if the particles are represented by meshes—notably if no constraints on convectivity are made and if explicit time stepping stops us from modelling complex particle shapes as compound of simpler convex shapes subject to a non-decomposable constraint. Injecting meshes particles into DEM is a single node challenge.

We introduce a triangle-based collision detection scheme for DEM that supports particles of arbitrary triangle count, configuration and size. Geometric comparisons suffer from poor SIMDability if realised straightforwardly as they involve many case distinctions. We recast the geometric checks into a minimisation that falls back to classic geometric checks as emergency solver. This way, we obtain a collision detection algorithm that is both robust and can exploit wide vector registers (Section 3). Furthermore, it can be parallelised on multiple cores either by deploying the triangles among multiple cores or by handling sets of triangles (batches) concurrently (Section 4). Some numerical results in Section 5 highlight the potential of our approach on multicore nodes before a brief summary and an outlook detail the impact on future DEM codes.

2 The particle and collision model

We study media composed of particles of arbitrary size. Each particle $p_i \in \mathbb{P}$ is described by a set of triangles $\mathbb{T}_i$. We do not impose any constraints on the triangle layout such as convexity. Our algorithms of interest consist of an explicit time stepping loop with a time step size Δt. Per time step, it runs over all particle pairs and identifies where any particle pair collides with each other: we determine the contact points. Per contact point, we determine the arising forces on the involved particles. Once all forces for all particles are accumulated, we update the particle velocities and positions and continue. If the contact point detection identifies that particles are too close to each other it halves Δt. If no contact points are identified at all, it increases Δt by 10%.

Our contact model is based upon an ϵ environment around each particle and a weak compressibility model for this ϵ-area: Two particles are in contact as soon as they are closer than 2ϵ. Mirroring Minkowski sums, we may interpret each particle to be enlarged by a soft layer of width ϵ (Fig. 1). Particles are in contact with each other as soon as these soft areas penetrate. If two particles are in contact, the contact point is the point that is closest to the particles' surfaces. We do not support contact areas at the moment but multiple contact points per particle pair may exist. Each contact point is associated one outer normal vector n per involved particle. Though our particles themselves are rigid, we call $\epsilon - |n|$ between a contact point and the real particle surface the penetration depth.

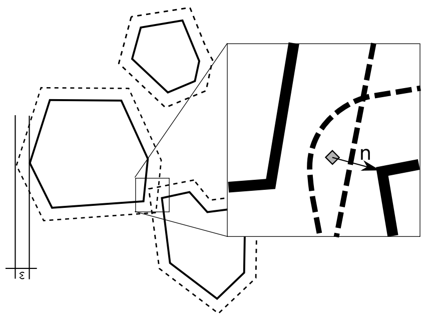

Fig. 1. Three particles with their ϵ environment. The particles do not penetrate each other, but two particles plus their ϵ environment penetrate and create one contact point (diamond point) with a normal n.

Our force computation equals pseudo-elastic damping as it is used in geometry overlapping methods [1]. We rely on the spring-dashpot DEM force model [3] which yields per particle pair p_i, p_j forces

$$f_{\perp}(p_i,p_j) = \begin{cases} S \cdot (\epsilon - |n_{ij}|) + 2D \cdot \left(\sqrt{\frac{1.0}{\frac{1.0}{m_i}+\frac{1.0}{m_j}}} \right) \cdot (v_{ij}, \frac{n_{ij}}{|n_{ij}|}) & \text{if } (v_i, v_j) \leq 0 \\ 0 & \text{otherwise} \end{cases},$$
$$f_{\|}(p_i,p_j) = r \times f_{\top}(p_i,p_j) \qquad (1)$$

acting on p_i. The forces on p_j result from parameter permutation. $(.,.)$ denotes the Euclidean scalar product, D is a damping, S the spring coefficient. v_{ij} is the relative collision velocity $v_j - v_j$. m denotes the mass of p_i or p_j respectively, n_{ij} is the contact normal pointing from the contact point in-between the particles onto the surface of particle j, i.e. from i to j.

The orthogonal force $f_{\perp}$ models solely repulsive forces, i.e. forces arise if and only if two particles continue to approach each other. The tangential force injects friction into the system. Obviously, system (1) is stiff and cannot avoid penetration of the real particles without their halo environment. We thus rely on small time step sizes Δt and reduce Δt as soon as $|n| \leq 0.2 \cdot \epsilon$ for any contact point normal in the system. $f_{\|}(p_i,p_j)$ is the torque force, where r is the lever arm of p_i's centre of mass to the contact point.

The plain algorithm is in $\mathcal{O}(|\mathbb{P}|^2 \cdot \mathbb{T}_{max}^2)$ with $\mathbb{T}_{max} = max_i|\mathbb{T}_i|$. We rely on a multiscale linked-cell approach based upon adaptive Cartesian meshes as it is used in molecular dynamics codes [4]: The computational domain is split into cubes that are at least as large as the biggest particle in the system and the cubes host the particles, i.e. hold links to the particles. The realisation stems from [11]. Particles can be in contact if and only if they are hosted by the same or neighbouring cells. This reduces the first quadratic term to a linear one as rigid particles cannot cluster arbitrarily dense. The second quadratic term results from the fact that we have to compare, for two particles p_i and p_j, each triangle from particle p_i with each triangle from p_j. Each pair of triangles requires fifteen checks: point-to-face $(2 \cdot 3 = 6)$ and edge-to-edge $(3^2 = 9)$. These comparisons are based upon a barycentric coordinate transform and yield a sequence of computations followed by if statements filtering out inadmissible solutions. The 15 distance computations then are reduced subject to the minimum function. Vectorisation of this approach labelled *brute force* suffers from branching and low arithmetic intensity.

3 A penalty-based, vectorising comparison method

With barycentric coordinates a, b, c, d over two triangles T_i and T_j that span a vector $x_i \in T_i$ and $x_j \in T_j$, we can cast the minimal distance problem into

$$min_{a,b,c,d} |x_i(a,b) - x_j(c,d)|^2$$

subject to the six inequality constraints $-a \leq 0, -b \leq 0, a + b - 1 \leq 0, -d \leq 0, -g \leq 0$ and $g + d - 1 \leq 0$. We refer to them as $c_1, \ldots, c_6 \leq 0$.

Our *penalty method* adds a Lagrange multiplier with $\alpha \cdot \sum_{i=1\ldots6} max^2(0, c(x_i))$ to the minimisation's objective function. It is a pseudo-transient continuation to penalise any solution out of the admissible region. The minimisation can not be passed to a plain Newton iteration as its Hessian is singular inside the admissible region, i.e. for valid solutions of a, b, c, d. Therefore, we resort to quasi-Newton where the Hessian is added an additional diagonal part $\delta \cdot id$, where δ is a tuned regularisation parameter and id is the identity matrix.

The penalty problem comes along with pros and cons. Its iterative character implies that there is no inner branching—*max* is supported by AVX—and it is arithmetically intense. In return, its performance is subject to two magic variables (α, δ) and there are always cases where it does not converge within a reasonable number of iterations for a chosen variable pair.

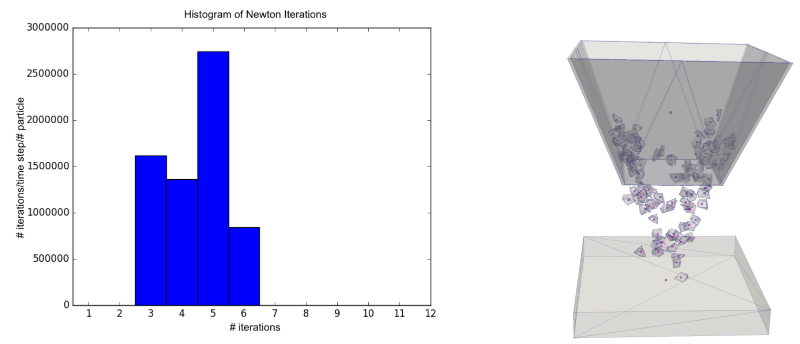

Fig. 2. Left: Histogram of Newton iterations required to solve characteristic triangle configurations. Right: Non-spherical granular material in a hopper setup.

A hierarchical collision check. Empirical studies suggest that the Newton iteration converges within few iterations for the majority of all of our meshes' triangles (Fig. 2). We thus propose a hybrid algorithm combining penalty and brute force. It fuses the iterative performance and brute force robustness.

Per triangle pair of two particles, our approach runs it_{Newton} Newton iterations. it_{Newton} is given. An epilogue then evaluates $c_1, \ldots, c_6$, i.e. the Euclidean norm (error) over $(max\{0, c_i\})_i \in \mathbb{R}^6$. If one constraint is harmed and the norm big, we trigger the brute force comparison variant.

SoA flattening of the triangle data structures Triangle meshes are logically hierarchical information consisting of triangles referencing spatial positions. To be able to exploit SIMD facilities efficiently, i.e. to avoid indirect memory access, our data structures replicate the vertex information and serialise the meshes. Our particle meshes are given as a sequence of vectors. Every three vectors in a row represent one triangle. On top of the actual geometric data, a hull struct holds all particle properties such as velocities, rotation, mass, geometric centre and mass centre, but also references to the vector sequences. The vectors of a vertex that is adjacent to k triangles thus is replicated k times.

In exchange for the memory increase, we can store the whole mesh as a structure of array [9] over the x, y and z coordinates with aligned arrays of `double`. Once all forces become available, our rigid body models determine translational and rotational updates. They are applied to all vertices. It thus is computationally acceptable to have vertex data replicated. It is automatically kept consistent as a particle mesh topologically does not change.

4 Shared memory parallelisation

Our multithreaded collision detection code runs a classic data decomposition scheme on the triangles: While the first triangle T_i of p_i is compared to the first triangle of T_j, we can simultaneously compare the second triangle. The concurrency scales in the number of triangles per particle. Synchronisation points, i.e. critical sections, are solely the insertion of contact points into the result set.

Further to our straightforward parallelisation of the hybrid collision model, we propose a batched multithreaded variant (Alg. 1). We make the hybrid collision checks exploit the multi-threaded environment by splitting the computational workload into groups of batches. A batch is a set of triangle pairs taken sequentially from the vertex vector arrays. It represents a subset of the triangles of a particle. Batches are empirically chosen to hold eight triangles in our experiments. As we know the size of a batch, we can collapse the Newton iteration loops and the batch iteration loop once we fix the number of Newton iterations. This widens the number of arithmetic operations subject to compiler reordering and vectorisation. The convergence check on $c_1, \ldots, c_6$ is then performed on a per-batch basis, i.e. we determine whether one of the triangles from T_i harms the admissibility condition. If this is the case, we apply our recursive scheme falling back to brute force to the whole patch.

Algorithm 1 Simplified hybrid contact detection fusing penalty-based and brute force checks into one algorithm. It is parallelised patch-wisely. `penalty` and `bf` are the penalty and brute force subroutine calls which are inlined. We have removed the recursion of the penalty blueprint here and fall back to brute force directly if an admissibility constraint is harmed.

```
for $k_i, k_j$=0; $k_i, k_j$<noOfTriangles/batchSize;k++ do
    OMP PARALLEL FOR REDUCTION (+:batchError)
    for $l_i, l_j$=0; $l_i, l_j$ <batchSize; l++ do
        $id_i$=$k_i$*batchSize + $l_i$; $id_j$=$k_j$*batchSize + $l_j$;
        distance[i] = penalty( $\mathbb{T}_i[id_i]$, $\mathbb{T}_j[id_j]$, error,$it_{Newton}$ = 4)      ▷ inlined
        batchError += error
    end for    ▷ Nested loops can be unrolled and reordered, as $it_{Newton}$ = 4 fixed
    if batchError/batchSize> $10^{-8}$ then                ▷ max reduction works, too
        OMP PARALLEL FOR
        for $l_i, l_j$=0; $l_i, l_j$ <batchSize; l++ do
            $id_i$=$k_i$*batchSize + $l_i$; $id_j$=$k_j$*batchSize + $l_j$;      ▷ Rerun with
            distance[i] = brute_force( $\mathbb{T}_i[id_i]$, $\mathbb{T}_j[id_j]$ )      ▷ robust algorithm
        end for
    end if
end for
```

It is usually not possible to predict the existence and distribution of "non-convergent" triangle pairs. Furthermore, the batch size choice is a tuning parameter. It determines the computation assigned per thread in terms of number of vectorised triangle pairs and thus correlates to OpenMP's chunk size [2]. The batch size also affects SIMD performance and penalty robustness. The larger the size the more triangle comparisons can be fused into SIMD statements. However, if one or more triangles fail convergence, the whole batch falls back to brute force. In theory, dynamic scheduling should resolve imbalances and thus mitigate effects of unwise batch size choices and unfortunate geometric constellations. Yet, experiments suggested that dynamic balancing does not yield a performance improvement; possible due to its overhead. Static scheduling is sufficient.

5 Results

Our code is a C/C++ collection of plain functions and structures. It is augmented by Intel SIMD and OpenMP pragmas which allows us to exploit SSE and AVX2 instruction sets. Results were obtained on an Intel Sandy Bridge 2.0GHz i5 node with 16 cores where we use Likwid [10] to read out hardware counters. Further experiments were conducted on Intel Xeon E5-2650 v4 (Broadwell) nodes with 24 cores each that run at 2.20 GHz. All result codes pass through Intel 15 (Sandy Bridge) and 17 (Broadwell) compilers. With respective code annotations the GNU compilers yield comparable yet slightly inferior throughput.

Single core hardware characteristics. We start with comparisons of the single node throughput of the plain brute force and penalty approach against the

STREAM Triad benchmark [8]. In this context, we run the code with and without vectorisation. As we refrain from intermixing penalty and brute force version into a hybrid, we let the penalty version converge up to a precision of 10^{-8}, i.e. the number of Newton iterations is not constrained. The measurement picks up a characteristics particle-to-particle comparison where one particle consists of 40 triangles.

Table 1. Hardware counter results for characteristic single-core runs on the Sandy Bridge chip. BF means brute force.

Metric	**Stream**	**BF**	**Penalty**	**BF+SIMD**	**Penalty+SIMD**
Runtime (s)	6.71	22.49	15.47	7.78	4.54
MFLOPS/s	1,245	962	1,073	2,808	3,202
CPI	0.48	0.48	0.49	0.91	1.26
Bandwidth MB/s	14,120	408	579	902	1,424

We observe (Table 1) that both variants do not exploit by any means the bandwidth that is available. This is promising w.r.t. the vectorisation. Indeed, the throughput triples almost for the vectorised iterative scheme. The brute force variant can not increase the throughput that significantly. Yet, we observe that both schemes benefit from SIMD—a fact also suggested by the compiler's vectorisation reports—which materialises both in an increase of cycles per instruction (CPI) and achieved MFLOPS/s.

Overall, the penalty-based version clearly outperforms the brute force variant which suffers from branching. We can expect the speed gap to widen once we fix or restrict the number of Newton iterations or supplement wider vector registers. Our results reveal furthermore that the algorithmic baseline is promising w.r.t. multicore parallelisation as the codes require a lower bandwidth than our STREAM baseline. We are not memory-bound.

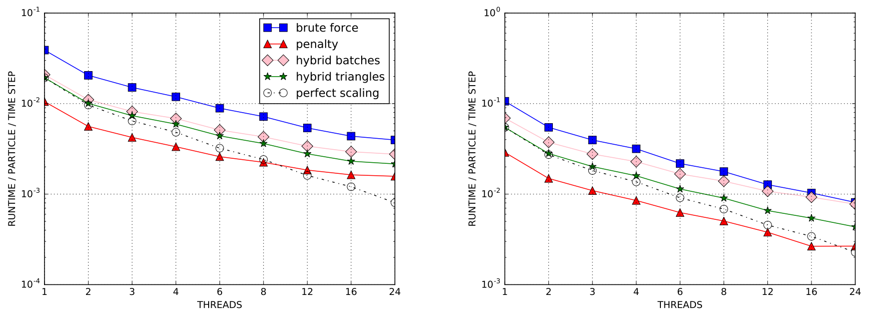

Fig. 3. Scaling of the various methods on the Broadwell. The setup runs a hopper scenario. Particles with 20 (left) or 40 (right) triangles each squeeze through a chute.

Time-to-solution without batches. A combination of the penalty variant with brute force as fallback yields a hybrid version that is robust, i.e. always gives a valid result. We continue with the hopper experiment where 1,000 particles are arranged in a regular Cartesian layout and fall down into a chute (Fig. 2). The code employs a grid technique but does itself run serially. Only for the basic particle-to-particle comparisons, it uses multithreading. We make the particles have roughly the same diameter but assign them randomised shapes consisting of 20 or 40 triangles. The penalty method tries four Newton iterations and afterwards directly falls back to brute force. We also present the scaling of the iterative method running at most four Newton steps. If we would not enforce termination after four steps, the iteration-based method would neither yield valid data nor be competitive with the other approaches due to outliers.

The hybrid's performance ends up in-between the performance of its two ingredients (Fig. 3). Our batched version is slower than a parallelisation based upon triangle decomposition. We however already observe that the gap between the two variants closes if we reduce the triangle count. Eventually, if one object consists of less than 10 triangles—such a situation occurs if a particle collides with the hopper geometry, e.g., as we model the hopper as set of independent, non-moving triangles—the batched version becomes faster (not shown). Overall, our algorithms exhibit reasonable scaling at least on one socket.

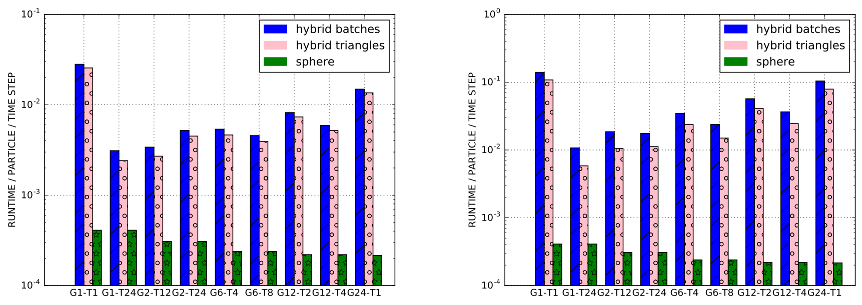

Fig. 4. Setup from Fig. 3 where the triangle and the grid cells and particle pairs run concurrently. Experiments on Broadwell. The `G` postfix describes how many threads are invested on the grid parallelisation while the digit following `T` describes how many threads are made available to the parallelisation of the collision checks.

Context While our approach successfully makes triangle-based particle collision exploit multicore processors, we have to assume that a sole thread decomposition of the triangle loops is insufficient. However, it is straightforward to combine classic mesh-based parallelisation with our approach: We decompose the grid into cells that are bigger than the particle diameter. A particle's triangle then is compared to the triangles of another particle if and only if the particles reside in the same or neighbouring cells. Cell pairs can be handled in parallel. Inside the parallelised loop over cell pairs, we also parallelise the collision checks over particle pairs. If three particles A,B and C are held in two neighbouring cells, we can deploy the three collision checks (A vs. B, A vs. C, B vs. C) to three threads. Inside this nested concurrency, we place our triangle-pair checks.

To contextualise our timings, we have rewritten our code to support spheres—the most convenient analytical shape. The tests continue with 1,000 spheres plus a grid [11] which eliminates roughly 90% of the potential collision checks a priori: including the sphere-hopper boundary checks, we run around $1.11 \cdot 10^5$ collision checks per time step instead of the $1.35 \cdot 10^6$ checks required without any grid. The sphere-based code basically streams data through the machine.

Our data uncover the significant speed reduction when we switch from spheres to meshed particles (Fig. 4). We "loose" one order of magnitude due to improved physics. With $\mathbb{T}_{max} \in \{10, 40\}$, our triangle-based approach furthermore yields a memory footprint that is almost by a factor of $9 \cdot \mathbb{T}_{max}$ `sizeof(doubles)` bigger than its spherical counterpart. For our small-scale setup, we observe that classic mesh-based parallelisation does not yield massive speedups. In contrast, our collision check parallelisation continues to speed up the computation by a factor of more than ten. An analysis of why the hybrid of grid and particle mesh parallelisation does not collaborate is subject of future research—reasons might be inappropriate pinning, strong scaling effects (too few particles) or memory boundedness. Yet, our results highlight how important a proper parallelisation of the most inner loops of the algorithm is.

6 Conclusion

Our manuscript introduces a collision detection algorithm for triangle-based particles that is able to exploit modern compute nodes both w.r.t. vectorisation and shared memory parallelisation. The key ingredient of our algorithm is to combine a weak collision model solved by an iterative scheme with a constraint on the iteration count plus a classic if-based collision check as fallback solver if the Newton iterations do not converge quickly. Depending on the type of object-object collision, we sketch a batched and a non-batched variant. Our code is one building block to obtain physically more reliable simulations as complex interaction functions mitigating non-spherical particle shapes can be replaced by first principle mechanics plus complex particle shapes. Furthermore, it demonstrates that triangle-based collision detection scales on multicore nodes. In particular, it scales the better the more accurate the geometric model.

Our results suggest that future manycore architectures will enable engineers to replace sphere-based DEM codes with software that relies on triangle meshes, as it has orthogonal characteristics to classic DEM codes—it is not bandwidth-bound, exploits vector registers and increases the code concurrency—and preserves DEM's spatially localised neighbour-to-neighbour checks. A showstopper for future research however is its massively increased memory footprint.

Our current and future work is three-fold. First, we embed our triangle-based comparisons into real-world engineering setups and study the qualitative impact of the non-spherical particles on the simulation outcomes. Second, we plan to extend our collision models to face contacts and to provide well-suited collision point postprocessing. One challenge with triangle-based collision detection results from the fact that multiple contact points can arise per particle pair. An algorithm has to decide which points from this set are duplicates as whole faces are near and parallel to each other. At the moment, we drop contact points that are closer to each other than the minimum mesh width of a particle mesh; which is a working yet physically unmotivated approach. Finally, we have to study how our approach integrates into distributed memory parallelisation. While most ingredients here are well-understood, it will be interesting to see how the SoA technique and MPI interfer. Our plan is to keep particles as atomic units and to work with whole ghost particles.

References

1. J. M. Boac, R. P. K. Ambrose, M. E. Casada, R. G. Maghirang, and D. E. Maier, *Applications of Discrete Element Method in Modeling of Grain Postharvest Operations*, Food Engineering Reviews, 6 (2014), pp. 128–149.
2. B. Chapman and J. LaGrone, *OpenMP*, Springer US, Boston, MA, 2011, pp. 1365–1371.
3. P. Cundall and O. Strack, *Discrete numerical model for granular assemblies.*, Geotechnique, 29 (1979), pp. 47–65.
4. M. Griebel, S. Knapek, and G. Zumbusch, *Numerical Simulation in Molecular Dynamics*, Springer, Berlin, Heidelberg, 2007.
5. K. Iglberger and U. Rüde, *Massively parallel granular flow simulations with non-spherical particles*, Computer Science - Research and Development, 25 (2010), pp. 105–113.
6. J. B. Johnson, A. V. Kulchitsky, P. Duvoy, K. Iagnemma, C. Senatore, R. E. Arvidson, and J. Moore, *Discrete element method simulations of Mars Exploration Rover wheel performance*, J. Terramech., 62 (2015), pp. 31–40.
7. T.-Y. Li and J.-S. Chen, *Incremental 3D collision detection with hierarchical data structures*, Proceedings of the ACM symposium on Virtual reality software and technology 1998 - VRST '98, 1998 (1998), pp. 139–144.
8. J. D. McCalpin, *Memory bandwidth and machine balance in current high performance computers*, IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter, (1995), pp. 19–25.
9. X. Tian, H. Saito, S. V. Preis, E. N. Garcia, S. S. Kozhukhov, M. Masten, A. G. Cherkasov, and N. Panchenko, *Effective SIMD Vectorization for Intel Xeon Phi Coprocessors*, Scientific Programming, 2015 (2015).
10. J. Treibig, G. Hager, and G. Wellein, *LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments*, in Proceedings of the 2010 39th International Conference on Parallel Processing Workshops, ICPPW '10, IEEE Computer Society, 2010, pp. 207–216.
11. T. Weinzierl, B. Verleye, P. Henri, and D. Roose, *Two particle-in-grid realisations on spacetrees*, Parallel Computing, 52 (2016), pp. 42–64.