A Contact Detection Code using Triangles for Non-Spherical Particle Simulations

Konstantinos Krestenitis[1]

March 5, 2016

[1]School of Engineering and Computing Sciences, University of Durham, DH1 3LE, Durham
konstantinos.krestenitis@durham.ac.uk
Mechanics Research Group

Supervised by
Dr Tobias Weinzierl
Dr Tomasz Koziara (former)
Professor Jon Trevelyan

Contents

1 Introduction

We present a Discrete Element Method (DEM) contact detection code that simulates rigid non-spherical particles on manycore shared memory machines and distributed memory computers. DEM is used to study granular particles in fields like soil mechanics. We rely on triangulated particle meshes to model surfaces. Spherical or multi-sphere models currently are state-of-the-art - due to a lack of well-suited software and runtime demands. Non-spherical particles promise to facilitate more accurate physics than sphere-based approaches [1]. The focus on triangles for rigid contact mechanics facilitates memory layouts allowing vectorised computation [2, 40, 41]. It is vital to yield high performance on current and upcoming processor architectures to enable engineers to simulate more realistic materials.

It is important to investigate the problem of finding the minimum distance between triangles because of the changes in the computational hardware [9]. The central processing unit architecture today and the future upcoming hardware support Single Instruction Multiple Data (SIMD)[2] parallelism which allow data level parallelism. These speed-ups are enabled because of new instruction sets and wide vector register. Furthermore, shared memory parallelism at the node level as well as distributed memory for supercomputers can provide significant speed-ups. It is vital to extract resources available on current and upcoming hardware, the speed-ups enable more triangles to be used to describe surfaces. Consequently, our objective is to enable engineers to use an algorithm that is capable to handle the maximum amount of triangles per time step to do better engineering and science.

In the present work we propose a hybrid method that combines the advantages of two optimised triangle-to-triangle distance computation methods. It benefits from both performance and robustness of an iterative Newton-penalty and a brute force solver. We exploit shared memory parallelism and SIMD (Single Instruction Data) to perform contact detection at the node level. On distributed memory, we use spatial domain decomposition whilst with the Message Passing Interface (MPI), we exploit asynchronous non-blocking communication to overlap data exchange with computation.

The state-of-the-art large scale DEM work, to the best of my knowledge, relies on sphere-based or multi-sphere particles [21, 22]. Large scale applications of spherical particles can be found in the field of molecular dynamics. Large scale non-spherical particle based DEM simulations in literature [20, 29, 28, 30] often describe particles as convex polyhedra and contact detection is resolved with GJK (Gilbert-Johnson-Keerthi) [14] variants which require interpenetration and access to all vertices of the particle to detect contact on the surface. Interpenetration introduces contact point clustering and the problem of divergence [38, 37, 41]. In addition, convex polyhedra based contact detection methods require undeterministic memory access to all particle simplices [14] which is contradictory to the aligned memory access imposed by SIMD for data locality. Triangle-based contact detection retain locality for vectorisation as well as increases accuracy of contact point generation [41]. Furthermore, the method can be extended to run on shared memory manycore and distributed memory machines.

The remainder of the paper is structured as follows. In Section 2, we describe the serial DEM algorithm. In Section 3 we review two triangle-to-triangle distance algorithms.

We create a new hybrid shared memory method that benefits from fast convergence and robustness. Next, in Section 3, we propose a distributed memory algorithm for contact detection that use asynchronous communication to overlap computation over communication. Section 4, discusses the future research directions.

2 Algorithmic Overview

- Completed since last report: Implemented serial DEM simulation with triangle-to-triangle contact detection using spring-dashpot contact forces, explicit time step.
- New concepts: New shared memory hybrid method for computation of triangle-to-triangle distance.
- Open questions: Best-case design of distributed memory implementation is not clear yet, real-world experiments are missing.

In contact mechanics, fluid-stucture interaction or other fields, it is an essential task to compute the distance between geometries to determine contact. Contact detection also is the most expensive algorithmic step [39, 57]. We present a Discrete Element Method (DEM) code that simulate particles that interact with spring-based contact. Non-spherical particles promise to facilitate more accurate physics than sphere-based approaches [1, 20]. The contact detection routine determines contact based on the distance between the triangles of every particle against all other. If two triangles are closer than a prescribed threshold, they are considered to contact each other. The use of triangles for rigid body contact dynamics rather than arbitrary polygons simplifies geometric checks and facilitates memory layouts that allow vectorised computation [13, 32, 41, 42].

The serial DEM Algorithm 1 demonstrates the execution of contact detection, an $O(n^2)$ operation. In line 3 and line 4 the nested for loop indicate the complexity of iterating through all triangle pairs. In line 5 TTD(i, j) invokes a Triangle-to-Triangle Distance (TTD) algorithm that determines the distance between triangle i and triangle j. Based on the boundary layer margin contact model [41] to avoid penetration, we check if distance between Triangle i and j is smaller than a set margin (line 6). If the parent particle of triangle i and j is not the same then this indicate a contact point between the two particles (line 7). The contact point information at line 13 and line 14 is used to derive the forces. The forces accelerate the particles that are then integrated by an explicit time stepping scheme such as explicit Euler.

Algorithm 1 DEM Serial Simulation Pseudo code

```
1 FOR time = 0; time < simulation time; time+=step
2   //contact detection
3     FOR i = 0 to N triangles
4       FOR j = i+1 to N triangles
5         distance = TTD(i,j)
6         IF (distance < margin) AND ParticleID(i) != ParticleID(j)
7           contact(PID(i)).add(point, normal)
8         ENDIF
9       ENDFOR
10    ENDFOR
11  //force derivation
12    FOR z = 0 to NB particles
13      FOR k = 0 to contacts(z).size()
14          force = granular(velocity(z), position(z), contacts(z).getcontact(k))
15      ENDFOR
16    ENDFOR
17  //explicit time stepping
18 ENDFOR
```

Algorithm 2 Spring-dashpot force algorithm

```
0 FUNCTION force = penaltyForce(normal, relativeVelocity, depth, massA, massB)
1  mass = 1.0 / ((1/massA) + (1/massB));
2  velocitymagnitude = DOT(relativeVelocity, normal);
3  magnitude = SPRING*depth + DAMPER*sqrt(SPRING*mass)*velocitymagnitude;
4  force = magnitude*normal;
5 ENDFUNCTION
```

Algorithm 2 shows the spring-based force derivation approach [38, 53, 57, 59]. To define contact points without penetration, we use the boundary layer margins to extend the surface boundary. Margin size is set based on velocities and time step size. When in contact, the margin overlap is used as the interpenetration depth. The contact point is at the middlepoint of the distance and normal direction is either side of the distance. At line 0, given the contact point normal, relative linear velocities, penetration depth and the mass of the two bodies we derive the interaction force to update the position of particle triangles on every time step.

3 Vectorised Contact Detection

- Completed since last report: Implemented serial and distributed memory robust contact detection and contact point generation.
- New concepts: Created new hybrid algorithm for triangle-to-triangle distance calculation.
- Open questions: Grain size and implications on shared memory load balancing.

We overview the brute force and penalty methods and we showcase the development of a new shared memory hybrid approach that benefits from both worlds. Both methods are implemented with two programming languages (MATLAB, C++). We focus on efficiency and we carefully benchmark the fragments on state-of-the-art architectures with respect to run-time. We review two triangle-to-triangle distance algorithms, the brute force and the penalty method. The two methods form the basis of the hybrid method that we propose.

3.1 Brute Force

The brute force is the naive approach to contact detection, it computes all distances between geometrical primitives of the triangles. The method computes the distance between all line segments of one triangle with the line segments of the other using the segment-to-segment (Algorithm 3, line 22-30) distance function [14, 56]. Then for every point of one triangle it computes the distance against the other triangle using the point-to-triangle (Algorithm 3, line 3-8) function [10, 14]. The method requires evaluating the minimum distance among all comparisons performed to find the overall minimum distance in the end.

The algorithm is straight-forward because it relies on only two geometric comparisons. As seen in Algorithm 3 in lines 3 to 21 the six comparisons yield one that defines the distance. For segment-to-segment, let v_{1i}, e_{1i}, v_{2i}, e_{2i} be the vertices and edges of triangles T_1 and T_2. The brute force approach computes all fifteen distances v_{1i}-to-T_2, v_{2i}-to-T_1, e_{1i}-to-e_{2i}, and then we select the shortest distance. As seen in Algorithm 3 in line 22 to 37 the algorithm determines the segment-to-segment distance between all segments of the triangle pair. Finally the algorithms finds the minimum of the two sets of comparisons to get the P and Q vertices that define the distance between two triangles.

Optimization of the brute force method can at most rely on avoiding redundant evaluations parts of the computation. It is not possible to avoid logical branching because we have to evaluate all if statements to arrive to the solution. Therefore it is not possible to optimize the method for SIMD parallelism because of the branching.

Algorithm 3 MATLAB Brute Force Solver.

```
0  FUNCTION [min,P,Q] = bf(A,B,C, D,E,F)
1  T1=[A;B;C;];  T2=[D;E;F;];
3  [ptlist(1), P0(1,:)]= pt([D;E;F], T1(1,:));
4  [ptlist(2), P0(2,:)]= pt([D;E;F], T1(2,:));
5  [ptlist(3), P0(3,:)]= pt([D;E;F], T1(3,:));
6  [ptlist(4), P0(4,:)]= pt([A;B;C], T2(1,:));
7  [ptlist(5), P0(5,:)]= pt([A;B;C], T2(2,:));
8  [ptlist(6), P0(6,:)]= pt([A;B;C], T2(3,:));
9  ptmin = min(ptlist);
10  FOR i=1:6
11    IF(ptmin==ptlist(i))
12      IF(i¡4)
13        p1 = T1(i,:);  p2 = P0(i,:);
15      ELSE
16        p1 = P0(i,:);  p2 = T2(i-3,:);
18      END;  break;
20    END;
21  END;
22  [ss(1),sp1(1,:),sp2(1,:)]=segseg(T1(1,:),T1(2,:),T2(1,:),T2(2,:));
23  [ss(2),sp1(2,:),sp2(2,:)]=segseg(T1(1,:),T1(2,:),T2(2,:),T2(3,:));
24  [ss(3),sp1(3,:),sp2(3,:)]=segseg(T1(1,:),T1(2,:),T2(3,:),T2(1,:));
25  [ss(4),sp1(4,:),sp2(4,:)]=segseg(T1(2,:),T1(3,:),T2(1,:),T2(2,:));
26  [ss(5),sp1(5,:),sp2(5,:)]=segseg(T1(2,:),T1(3,:),T2(2,:),T2(3,:));
27  [ss(6),sp1(6,:),sp2(6,:)]=segseg(T1(2,:),T1(3,:),T2(3,:),T2(1,:));
28  [ss(7),sp1(7,:),sp2(7,:)]=segseg(T1(3,:),T1(1,:),T2(1,:),T2(2,:));
29  [ss(8),sp1(8,:),sp2(8,:)]=segseg(T1(3,:),T1(1,:),T2(2,:),T2(3,:));
30  [ss(9),sp1(9,:),sp2(9,:)]=segseg(T1(3,:),T1(1,:),T2(3,:),T2(1,:));
31  ssmin = min(ss);
32  FOR i=1:9
33    IF(sslist(i) == ssmin)
34      csp1 = sp1(i,:);  csp2 = sp2(i,:);
36    END;
37  END;
38  min = min(ssmin, ptmin);
39  IF min == ptmin
40    P = p1;  Q = p2;
41  ELSEIF min == ssmin
42    P = csp1(:,:);  Q = csp2(:,:);
43  END;
```

3.2 Penalty

The second approach to solve the triangle-to-triangle distance problem is by parameterising of the triangles such that the distance between them is formulated by a quadratic function and a solvable minimization problem. This approach is iterative compared to brute force. The method approximate the solution using Newton iterations.

The method relies on an optimization-based formulation of the distance between two points. Let x and y be two points, belonging respectively to triangle T_1 and T_2. Assuming that points A, B, C are vertices of T_1 and that points D, E, F are vertices of T_2, x and y can be defined using the following equations:

$$T_1 : x(a,b) = A + (B - A) \cdot a + (C - A) \cdot b$$

$$T_2 : y(g,d) = D + (E - D) \cdot g + (F - D) \cdot d$$

To find the minimum distance between T_1 and T_2 and the corresponding two closest points P, Q on the two triangles we minimize

$$f\left(a,b,c,d\right) = \left\|x\left(a,b\right) - y\left(c,d\right)\right\|^2$$

What has to be noted is that x and y have to stay within the area of the two triangles. The four parameters of the function f have to comply with six inequality constraints:

$$min_{a,b,g,d} f(a,b,g,d)$$

$$such\ that:\ \{a \geq 0, b \geq 0, a + b \leq 1, d \geq 0, g \geq 0, g + d \leq 1\}$$

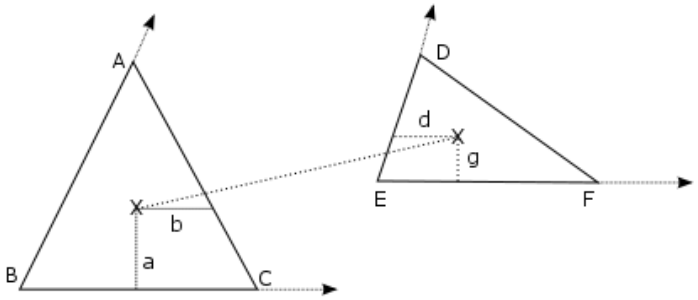

Figure 1: Example of minimum distance and the corresponding barycentric points (parameters of objective function) on a pair of triangles in 3D. Triangle X:T1 has points A, B, C where barymetric parameters a,b correspond to point x on the triangle. Triangle Y:T2 has points D,E,F where barymetric parameters g,d correspond to a point x. The two defined barymetric points define the minimum distance between the two triangles in 3D.

To find the minimum of the parabolic constraint problem the Newton-Raphson method is used. Newton method use information from the curvature of the problem using the Hessian and gradient to arrive to a minimum point. To enforce the constraints the problem

is transformed into a series of unconstrained problems with the augmentation of the objective function f using the penalty-based method.

The penalty function penalizes the iteration so that the boundaries of the feasible region are valid; at least in a weak sense. Although there are factors that determine the number of iterations required the tuned-to-the-problem parameters yield a good number of Newton iterations that cannot be reduced to less than two; (initial guess to penalized boundary step, correction step).

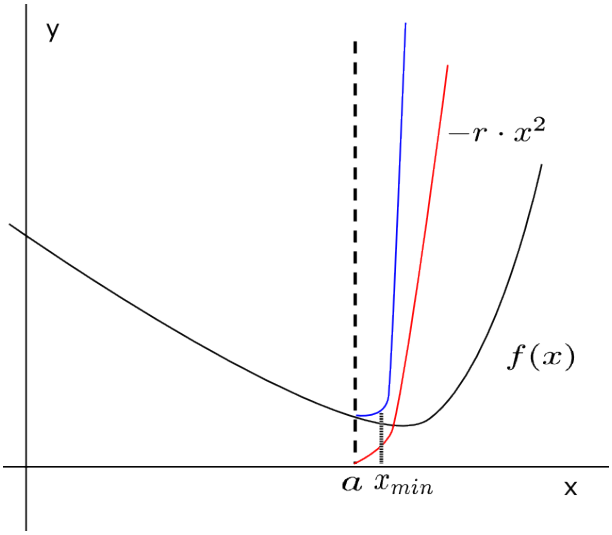

Figure 2: Illustration of a 2D problem showing the penalty function (red line) penalizing the objective function (black line) f(x) under a constraint a (dash line) to create the feasible region (blue line).

This iterative approach adds a penalty term to the objective function to penalize the solution when outside of the feasible region:

$$P(x) = f(x) + r \sum_{i=1...6} max(0, c(x_i))^2 \tag{1}$$

Where r is the penalty parameter (Figure 2). Newton iterations always converge to a solution slightly on the outside of the feasible region. Convergence can be controlled by the r penalty parameter that controls the sharpness of the curve for the constraints. One aspect that requires care however is the invertibility of the Hessian $\nabla\nabla P$.

Furthermore that the problem is ill conditioned and a Quasi-Newton method has to be used. The Hessian matrix is not invertible so it is not possible to solve the direction of the search by computing the Hessian and gradient. This illustrates the fact that f has multiple minima and $\nabla\nabla f$ is singular. Consequently, $\nabla\nabla P$ is also singular inside of the feasible region. The ill conditioning is caused by the problem definition itself, where there is a state where there are multiple solutions to the problem based on the orientation of the two triangles. This is also revealed by the two zero eigenvalues of the Hessian. Because of the ill-conditioning, we use a quasi-Newton approach, where the Hessian is approximated by a perturbed operator $\nabla\nabla P + \epsilon I$. I is an identity matrix and ϵ is suitably small.

Algorithm 4 MATLAB Penalty Solver.

```
0  FUNCTION x = penalty(A, B, C, D, E, F, rho, tol)
1  BA = B-A; CA = C-A; ED = E-D; FD = F-D;
2  hf = [2*BA*BA', 2*CA*BA',-2*ED*BA',-2*FD*BA';
3        2*BA*CA', 2*CA*CA',-2*ED*CA',-2*FD*CA';
4       -2*BA*ED',-2*CA*ED', 2*ED*ED', 2*FD*ED';
5       -2*BA*FD',-2*CA*FD', 2*ED*FD', 2*FD*FD'];
6  x = [0.33; 0.33; 0.33; 0.33];
3  FOR i=1:99
4    X = A+BA*x(1) + CA*x(2);
5    Y = D+ED*x(3) + FD*x(4);
6    gf = [2*(X-Y)*BA'; 2*(X-Y)*CA'; -2*(X-Y)*ED'; -2*(X-Y)*FD'];
7    h = [-x(1); -x(2); x(1)+x(2)-1; -x(3); -x(4); x(3)+x(4)-1];
8    dh = [-1, 0, 1, 0, 0, 0; 0, -1, 1, 0, 0, 0;
9           0, 0, 0, -1, 0, 1; 0, 0, 0, 0, -1, 1];
10   mask = h' >= 0;
11   dmax = dh.* [mask; mask; mask; mask];
12   gra = gf + rho * dmax * max(0,h(:));
17   hes = hf + rho*dmax*dmax' + eye(4,4)/rho^2;
18   dx = hes\gra;
19   DX = BA*dx(1) + CA*dx(2);
20   DY = ED*dx(3) + FD*dx(4);
21   error = sqrt(DX*DX'+DY*DY');
22   IF error < tol, BREAK; END
23   x = x - dx;
24 END
25 END
```

The penalty algorithm in pseudocode language is shown in Algorithm 4. It accepts A, B, C, D, E, F vector coordinates for triangle T1(A, B, C), T2(D, E, F) as well as the required parameters for the algorithm to be solved. Rho is the penalty parameter that controls the steepness of the P(x) function (equation 1) , eps is the perturbation parameter for the hessian matrix of the problem along its diagonal to make the matrix solvable. Tol is the tolerance for convergence (Floating point accuracy). In line 6 of Algorithm 4 an initial guess is chosen to be the center of the two triangles, then the for loop initiates the Newton iterations to find the points on the X, Y triangle planes under the constraints c. For each of the six constraints (line 12) the max function of the penalty is determined so that every possible active constraint is detected. In line 19 and line 20 the gradient and Hessian of P Penalty function is evaluated to be provided to the Gaussian elimination direct solver so that a Newton direction DX is solved. If the Newton step is large enough over the specified tolerance then the iteration is converged else the direction is used and the loop is executed once more recursively.

The penalty method is well-suited for SIMD optimisation. We can concurrently determine the distance between multiple triangle pairs as long as we use the same number of Newton steps: Up to four or eight triangle pair distances then are determined at the same time. Such a speed-up statement however has to be read carefully. While the concurrency is high, it is not clear a priori how many Newton steps are required. A high number of Newton steps can render the penalty method slower than a brute force approach.

3.3 Hybrid

We propose a new hybrid algorithm that combines both penalty and brute force to solve the distance problem $f(a,b,c,d)_{min} = \|x(a,b) - y(c,d)\|^2$. The method benefits both from iterative penalty-based approximation performance and brute force robustness. The algorithm switches from penalty to brute force on run-time when there is no convergence. In a modern CPU environment the hybrid method exploits the Simultaneous Multi-Threading (SMT) microprocessor architecture on top of Single Instruction Multiple Data (SIMD) parallelism [22].

We exploit the multi-threaded environment by splitting the computational workload into groups of batches. The batches are a set of triangle pairs that are selected sequentially in a memory aligned vector array. We fuse triangle comparisons with threads to exploit thread level parallelism and wide SIMD registers. Convergence checking is performed on a per-batch basis to determine fall-back to brute force.

Based on empirical studies and tuning of the penalty parameter and the regularization variable, the majority of triangle pairs are solved within four iterations as shown in Figure 3. When the batch is solved within four iterations then it runs a penalty solver but that is not always the case. The memory distribution of pairs of triangles that do not converge within four Newton iterations are not known a priori because the solution depends on the underlying geometry. Triangle pairs that do not converge skew the error margin of the batch creating the worse case scenario where the hybrid method becomes a brute force solver. It is not possible to predict the sparse distribution of non-convergent triangle pairs during run-time.

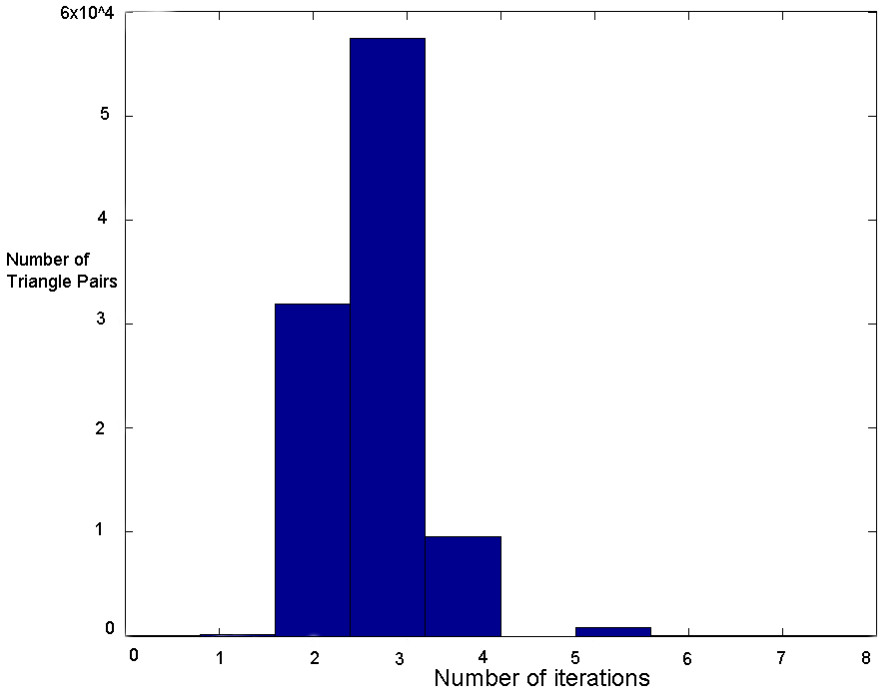

Figure 3: Histogram of iterations required to solve random pairs of triangles using penalty.

A tuning factor is the batch/grain size. The grain size is the computation assigned per thread in terms of number of vectorized triangle pairs. The grain size affects the error value and the SIMD performance performed by penalty and brute force. The larger the grain the more triangle comparisons can be fused into SIMD statements. However, if one or more triangle fail convergence, the whole batch may need to fall back to brute force. The grain size affects thread-level load balancing. in theory, dynamic scheduling should resolve imbalances. Dynamic scheduling uses the work queue to provide a dynamically sized block of loop iterations to each thread during run-time. This method creates an extra scheduling overhead to the algorithm, due to thread safety and other load balance auxiliary metrics. Experiments show that dynamic balancing does not yield performance improvement due to overhead. Static scheduling is sufficient. Tuning the grain is non-trivial due to the random distribution of triangle pairs in memory. Trial and error fine-tuning over a given set of random triangles work best for static load balancing. The batch size for random triangles is optimal at size eight.

Algorithm 5 Hybrid Contact Detection

```
1  n=sizeoftriangles
2  batchSize = 4;
3  error[batchSize]
4  batchError = 0.0;
5  FOR (k=0; k<n/batchSize; k++) //grain-size is four
6     batchError = 0.0;
7    PRAGMA OMP PARALLEL FOR SIMD REDUCTION (+:batchError)
8    FOR (l=0; l<batchSize; l++)
9      i=k*batchSize + l;
10       distance[i] = penalty(T1[i], T2[i], P[i], Q[i], error[l]);
11    ENDFOR
12    batchError = 0.0;
13    FOR (l=0; l<batchSize; l++)
14       batchError += error[l];
15    ENDFOR
16    IF(batchError/batchSize>1E-8)
17       PRAGMA OMP PARALLEL FOR SIMD
18       FOR (l=0; l<batchSize; l++)
19          i = k*batchSize + l;
20          distance[i] = bf(T1[i], T2[i], P[i],Q[i]);
21       ENDFOR
22    ENDIF
23 ENFOR
```

In Algorithm 5 we show the serial pseudocode of the hybrid penalty-brute force method. In line 2 we set the batchSize to four, that is the grain size. In line 6 we split the computational workload n into batches based on the grain size and loop through the batches. In line 9 we initiate a parallel threaded loop over SIMD operations within each batch and solve each triangle pair with penalty. Line 9 specifies the reduction instruction to perform a parallel sum reduction on the batch error per pair. In line 10 the penalty method is executed for one triangle pair T1[i] and T2[i] where P[i], P[i] are vertex points that define the three dimensional distance[i]. In line 14 the overall batch error is summed and in line 16 it is compared against a manually set tolerance. If batchError exceeds the tolerance then the algorithm calls brute force.

We implemented the new hybrid method in the C programming language with SIMD enabled on Intel Sandy Bridge 2.0GHz i5. In Figure 4 we showcase the measurements of shared memory hybrid approach against the penalty and brute force methods. As expected the hybrid performance is settled between brute force and penalty. The method is slower than penalty because there are batches of triangle pairs that do not reach the convergence tolerance. It is faster than brute force because no significant number of batches fail the tolerance. The hybrid algorithm scales over the second socket of the CPU but does not gain significant performance between eight and sixteen cores because of data stagnation at the socket interconnect. Theoretically, performance can be tuned further by investigating the load balancing implications on the non-deterministic element of the algorithm. The results are sufficient with static scheduling for proof of concept.

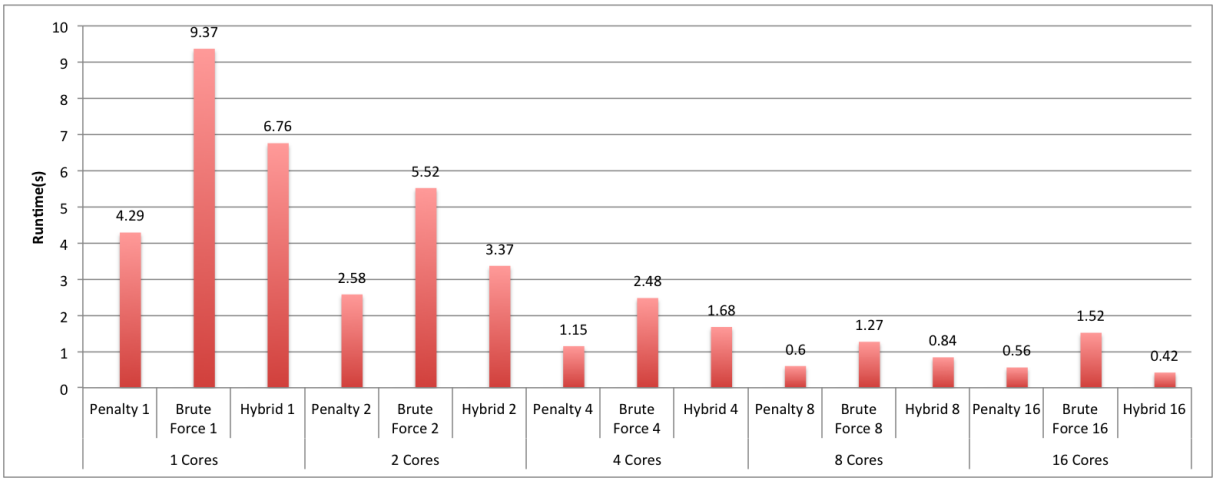

Figure 4: Run-time comparison of the penalty, brute force and the hybrid method over no of cores.

4 Domain Decomposition for Contact Detection

- Completed since last report: Implemented domain decomposition for distributed memory load balancing using library, MPI blocking communication for data migration.
- New concepts: Overlapping process communication for contact detection with ghost data. Asynchronous communication waiting time significant as number of ranks increase.
- Open questions: multilayer grid/spacetree implementation for DEM

We further scale the computation of triangle-to-triangle contact with the Message Passing Interface (MPI) [16] for distributed memory computation. Our DEM-based contact detection is inspired by molecular dynamics simulations, where millions of spheres are simulated. State-of-the-art MD computation is decomposed into smaller computational sub-domains and communication exchanges necessary boundary information. Similar computational stages are applied by us in the DEM-based contact detection algorithm. We study synchronous and asynchronous modes of data exchange and how two communication strategies can be exploited to increase performance and minimize computation.

The distributed memory DEM contact detection algorithm is performed in three stages. In stage one the computational domain is divided into equally balanced sub-domains. In stage two, data migration assigns the sub-domains to MPI instances. In stage three, we compute the distances to determine contact between all triangles. In stage four we accumulate the forces and in stage five we perform the explicit time step integration.

4.1 Domain Decomposition

To scale the performance of contact detection simulation, the computational workload has to be processed in parallel by splitting the domain into sub-domains and computation deployment is based on it, making the decomposition an important stage. Decomposition affects load balancing and the communication patterns of the simulation. Communication is essential when sub-domains are inter-dependent since new boundaries are introduced with decomposition. It is important to decompose evenly whilst also reducing sub-domain communication. There are three types of decomposition that are widely used in different contexts [11, 47] that can apply in DEM. Decomposition by particle, decomposition by force and spatial decomposition.

In domain decomposition by particle [37], the domain is divided such that all sub-domains hold equal number of particles. For N particles we split the domain to N parts and we assign them to P processors (N/P). The sub-domain boundary is always located at the space between particles. In this method an all-to-all communication is required to detect contact points because they are always located at the sub-domain boundary. The global communication is a major disadvantage for small particles as each processor communicates with all other processors, staggering the overall parallel processing. Another disadvantage is the non-equal splitting (N/P) as some particles may require more computation than others (i.e. more triangles). The main advantage of particle-based decomposition is the implementation simplicity [47].

An alternative decomposition approach found in literature is force-based decomposition that is often applied in molecular dynamics (MD) and smoother particle hydrodynamic (SPH) simulations [4, 19, 22, 47, 48, 50]. The method formulates a interaction force matrix that solves the contact detection and the forces as a group of linear equations. The matrix is decomposed into small blocks and shared on each process to solve in parallel. Force based decomposition assume short-range interactions and thus a dense force matrix to solve. In force based methods, the advantage is that computation is decomposed without any spatial information from the particles. The major disadvantage in DEM applications is the assumption of dense and uniform sparse force matrices, the assumption can be wrong leading to imbalanced domains on run-time.

According to N-body simulation literature [11, 15, 58, 60] the state-of-the-art method for supercomputing applications is the spatial decomposition. Our decomposition is based on the spatial position of vertices using Recursive Coordinate Bisection (RCB) [3]. In RCB, the computational domain is first divided into two regions by a cutting plane orthogonal to one of the coordinate axes so that half the work load is in each of the sub-regions. The splitting direction is determined by computing in which coordinate direction the set of objects is most elongated, based upon the geometric locations of the objects. Each triangle vertex thus is owned and persistently stored exclusively on one process/sub-domain for the whole duration of the timestep.

Spatial decomposition implicitly creates logical boundaries that split space and computation (c.f. Figure 5). The contact detection complexity of N trianglesthat belong to n_i particles without boundary boxes is $\sum_{i=0}^{n} n_i = \mathrm{O}(n^2)$ in big-O notation (Algorithm 1). Contrary, with equally spaced boundary boxes/cells, if C is the maximum number of particles

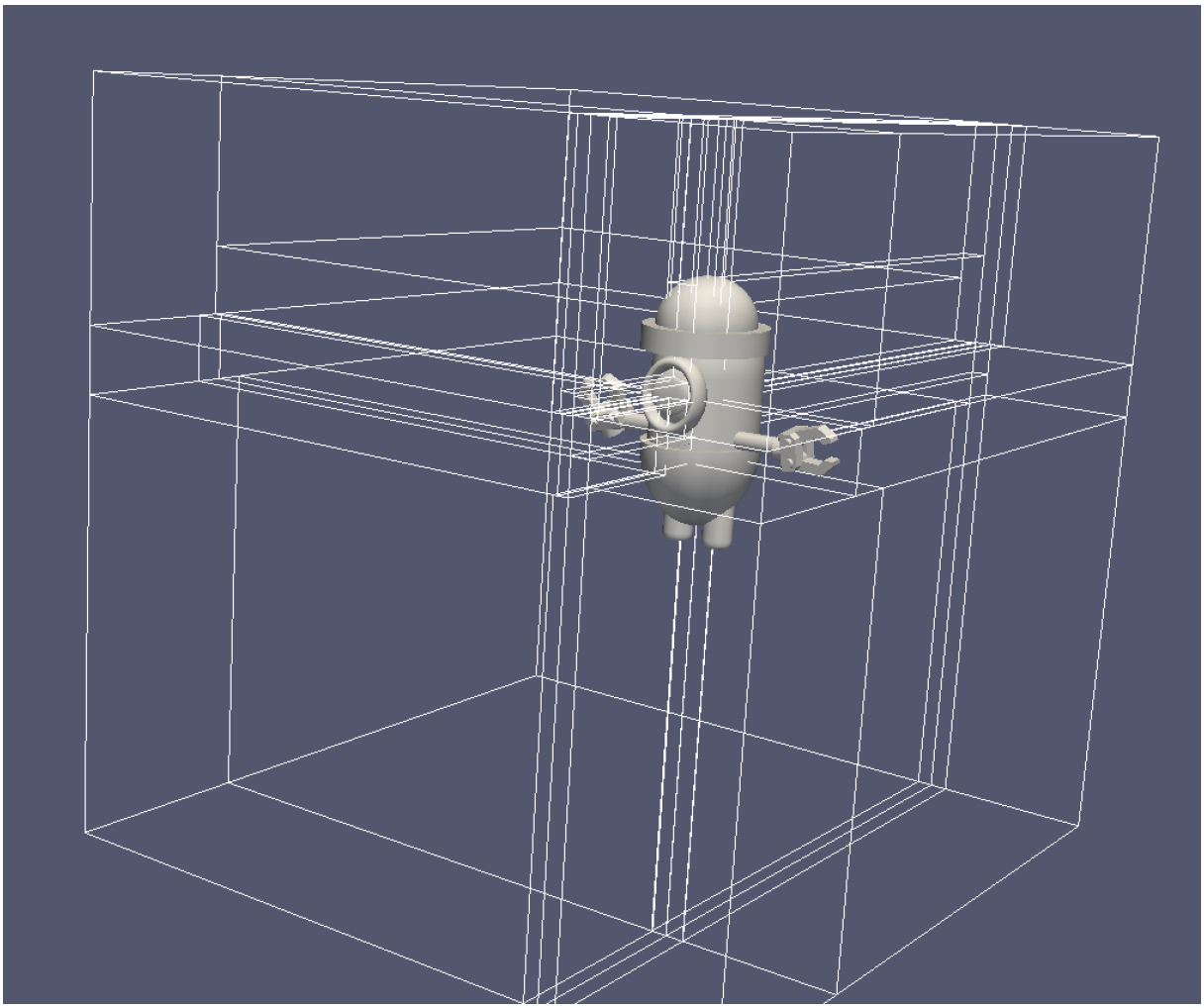

Figure 5: Spatial domain decomposition using triangular elements on 32 MPI processes, visible boundaries cut space/domain of each process.

then there are 26 neighboring cells and $max(n_i)$ is the maximum number of triangles per particle i which result to $n_1 \times 26 \times C \times max(n_i)$ triangles per particle i in the local cell neighborhood. For all N triangles in the domain there are $N \times 26 \times C \times max(n_i)$ per neighborhood. Boundary boxes due to spatial decomposition reduce the overall computational complexity to $O(n)$. Similarly, in MD simulations a cut-off range set the range of interaction fields per particle to remove redundant computation. Equally spaced boundary boxes rely on octree-based variants data structures that are great for recursive eight cell space subdivision. Our decomposition relies on non-uniform recursive subdivisions that rely on kd-tree-based data structure decomposition [6] (i.e. RCB) and the number of neighbours can be arbitrarily many as they are not equally spaced, but they are equally vertex-sized.

The number of neighbours for both methods may have an implication on communication patterns between processes. It is an open question what are the performance implications on uniform spaced octree-based decomposition versus the non-uniform kd-tree based decomposition. It is an area of investigation since on an octree-based approach information about the level of refinement is known a priori by the sub-domain boundary size, which

is an interesting application for multiscale simulations.

When spatial decomposition finishes we migrate the data to the processors (Algorithm 6 line 2) with blocking synchronous communication. At each time step the triangles migrate according to the DEM kinematics. In addition to migration, a local area data exchange is required to communicate the boundaries of the sub-domains that cut triangles at the boundary.

4.2 Ghost Data Exchange

The triangles that overlap into a remote sub-domain due to the decomposition cuts are temporary copied to one or more sub-domains in order to perform a complete contact detection. In this section, we study MPI communication characteristics when we resolve data dependencies between the sub-domains caused by ghost triangles in order to maximize performance of distributed contact detection. We explore the implication of inter-process communication and local computational performance using two communication strategies that use asynchronous non-blocking communication.

Algorithm 6 Naive Asynchronous Data Exchange Pseudocode

1. Load balance triangles
2. Migrate triangles to MPI network using blocking communication
3. Initiate neighbourhood all-to-all asynchronous MPI send/receive
4. Wait for neighbourhood asynchronous communication to terminate
5. Contact detection
6. Derive contact forces from contact points generated
7. Explicit time integration

The first strategy exchanges local data to all neighbours (Algorithm 6 line 3). The goal is to utilize the communication bandwidth while minimise communication administration overhead. If the exchange does not reach the upper bandwidth limit then exchange of all data is faster than filtering out the ghost triangles, i.e. doing any preprocessing that finds out which triangle from a sub-domain might be required from a neighbour. Alternatively, we can send out only triangles that overlap from one sub-domain into another sub-domain. This filtering of triangles is an $O(N)$ operation. As soon as MPI communication finishes, the algorithm invokes the existing contact detection routines exploiting vectorised floating point operations with a single contact detection routine. Exchange of all local data to all neighbours significantly increases the number of triangles to be processed from T_{local} to $(T_{local} * N_{ranks} * T_{remote})$ where N_{ranks} is the number of neighbours of neighbours. The increase of triangles to be checked increases the total computation performed locally because of the redundant triangles.

The disadvantage of such a naive method is that total MPI wait (figure 6) for an all-to-all neighbour data exchange increases with the number of processes. The asynchronous communication wait time results to time wasted with idle processors. The method is potentially useful with decomposition schemes where all data exchange can be processed in the background, i.e. enough bandwidth is available.

Figure 6: Waiting time (s) per MPI rank/node for all to all neighbour data exchange over 1000 time steps (25 mil triangles, 10k non-spherical particles).

Algorithm 7 Overlapping Asynchronous Data Exchange Pseudocode

```
1 Load balance triangles
2 Migrate triangles to MPI network using blocking communication
3 Search overlapping ghost triangles to send
4 Initiate neighbours asynchronous MPI send/receive
5       Local contact detection
6       Retrieve required ghost triangles from neighbours
7       Local against to remote ghost triangle contact detection
8 Wait for neighbourhood asynchronous communication to terminate (No Real Wait)
9 Derive contact forces from contact points generated
10 Explicit time integration
```

The second strategy filters out local ghosts from the data structure and sends them to the overlapping neighbouring processes. Using the spatial decomposition information, we find the specific processes/boundary cells that the triangle bounding box overlaps. In this strategy we minimize data exchange at the cost of a filtering overhead and the allocation of buffers in memory. The method aims to minimise the waiting time of MPI processes by overlapping concurrent computation over communication. As shown in Algorithm 7 line 5 local contact detection is executed as soon asynchronous MPI communication is initiated overlapping communication. A second contact detection is initiated to determine contact between local and remote ghost triangles at line 7. The strategy is advantageous as neighbour waiting is always zero as long as local contact detection takes longer than the transmission of the data. As shown in Figure 6 the waiting time is increased proportionally to the number of MPI ranks, enabling us to increase overlapping computation per process with larger number of processes.

We measure both strategies to determine the performance of normalised computation per triangle. The result in Figure 7 shows for each rank and compute node on the x axis, the time required to solve the distance of a pair of triangles. The blue bars show the average

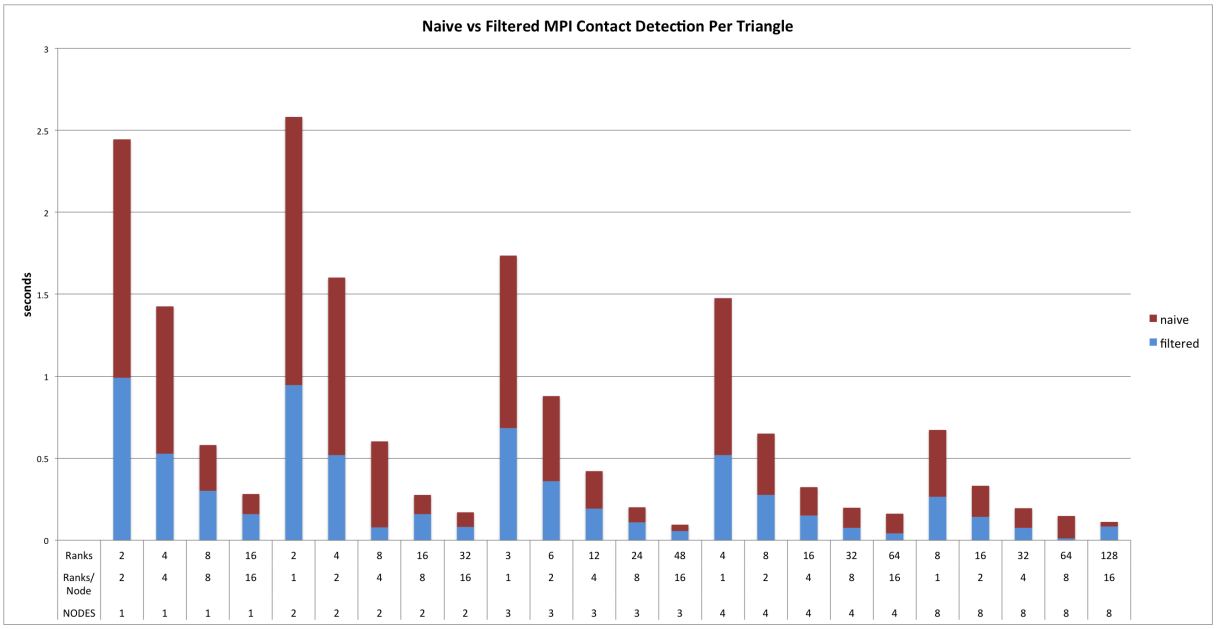

Figure 7: Normalized naive vs filtered contact detection performance per triangle pair running on multiple nodes.

normalised time for the second strategy to process a triangle (filtering approach). The red bars show the difference between the overall naive communication time and the filtered method. The time is reduced linearly as the number of ranks increase.

For the measurement set-up in Figures 6 and 7, we use Durham University Hamilton supercomputer that has per node 2 x Intel Xeon E5-2650 v2 (Ivy Bridge) 8 cores, 2.6 GHz processors, 64 GB DDR3 memory, 1 x TrueScale 4 x QDR single-port InfiniBand interconnect.

5 Conclusion and Future Work

We implement a new fast but also robust hybrid algorithm that exploits modern CPU architectures. In addition, we study different schemes of domain decomposition in literature. I implement the state-of-the-art spatial domain decomposition and load balancing using Zoltan. Manual migration has been implemented using blocking MPI communication. We implement an asynchronous communication for data exchange of ghost data and compare two possible strategies. Finally based on benchmark measurements we pick the best performing scheme that overlaps computation, increases performance and decreases data exchange volume.

The next stage of my research is to investigate contact detection strategies for large scale supercomputing on a multiscale setting using adaptive multiscale grids. I'm also interested in vectorized memory layout implications on MPI buffered communication. Lastly, the source code will be generalized to form a open source library that any simulation tool can use.

According to the project plan of the previous report, the work packages have been completed as projected.

5.1 Acknowledgments

This work has been sponsored by EPSRC (Engineering and Physical Sciences Research Council) and EDF Energy as part of an ICASE studentship. This work also made use of the facilities of N8 HPC provided and funded by the N8 consortium and EPSRC (Grant No. N8HPC_DUR_TW_PEANO). The Centre is co-ordinated by the Universities of Leeds and Manchester. We also thank University of Durham for the supercomputing resources and technical assistance. All underlying software is open source and available at: https://github.com/KonstantinosKr/delta.

5.2 Work Packages

WP 1. MPI force data exchange

- Benchmark force per particle accumulation for DEM simulation. When contacts are determined and forces derived, the forces are required to be communicated back the ranks that hold the center of mass to update the kinematics.
- Risk: Benchmarking and testing may take longer than March 2016.

WP 2. Total contact MPI Benchmark and code optimisation

- Further code optimisation for memory alignment and reducing cache misses benchmark on the total DEM code.

WP 3. Write journal paper on contact detection

- Report on contact detection work done over the last two years and produce a complete paper with the new insights.
- Risk: there is a possibility that the paper writing will delay research in the next two working packages.

WP 4. Create generic and accessible library (EDF Energy, Peano, Others)

- Create an open source modular library so that it is usable by others in industry and in science.
- Risk: spending more than enough time to do unnecessary non-scientific software engineering work. (eg. data input, visualization)

WP 5. Multiscale peano integration

- Integrate with the contact detection library into Peano framework. The goal is to use adaptive grid to create a multiscale DEM code.

- Risk: potentially setting up the contact detection library to be usable by the framework may lead to the creation of code that is application specific instead of generic.

WP 6. Write paper and journal on multiscaling

- Report on the new experiments with multiscale contact detection using the peano framework.
- Risk: getting new results from experiments on time to produce high quality paper on time. It is a risk that is linked the implementation time required.

[54]

5.3 Conferences

-Conference - Euro Par

- Autumn 2016
- Conference paper on DEM with load balancing. Link: http://www.europar2015.org/

-Conference - ACME 2017

- Spring 2017
- Complete code framework for large scale DEM simulations. Link: http://www.europar2015.org/

-Conference - International Supercomputing Conference

- May 2017
- Whole project research. Link: http://www.isc-events.com/isc14/

Gantt Chart - Work Packages												
	January	February	March	April	May	June	July	August	September	October	November	December
	2016											
Write 18 month PhD Report												
ACME Conference 2016												
SIAM PP Conference 2016												
Write Journal Paper on HPC DEM simulation												
Paralle Rigid Kinematics (Force Communication)												
MPI Code Optimization												
Library creation - Accessibility (EDF Energy)												
Integration with PEANO for multiscale simulation												
	2017											
ACME Conference 2017												
International Supercomputing Conference 2017												
Write Journal paper on multiscale simulation												
Write Thesis												

Figure 8: Timeline of Project

References

[1] Fernando Alonso-Marroquin, Alvaro Ramirez-Gomez, Carlos Gonzalez-Montellano, Nigel Balaam, Dorian A H Hanaor, E. A. Flores-Johnson, Yixiang Gan, Shumiao Chen, and Luming Shen. Experimental and numerical determination of mechanical properties of polygonal wood particles and their flow analysis in silos. *Granular Matter*, 15(6):811–826, 2013.

[2] Mauricio Alvarez, Esther Salami, Alex Ramirez, and Mateo Valero. Performance impact of unaligned memory operations in SIMD extensions for video codec applications. *ISPASS 2007: IEEE International Symposium on Performance Analysis of Systems and Software*, pages 62–71, 2007.

[3] E. G. Boman, U. V. Catalyurek, C. Chevalier, and K. D. Devine. The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: Partitioning, ordering, and coloring. *Scientific Programming*, 20(2):129–150, 2012.

[4] Kevin J. Bowers, Ron O. Dror, and David E. Shaw. Zonal methods for the parallel execution of range-limited N-body simulations. *Journal of Computational Physics*, 221(1):303–329, 2007.

[5] Robert Bridson, Ronald Fedkiw, and John Anderson. Robust treatment of collisions, contact and friction for cloth animation. *ACM Transactions on Graphics*, 21(3):594–603, 2002.

[6] Russell A Brown. Building a Balanced k-d Tree in O (kn log n) Time. 4(1):50–68, 2015.

[7] T.S. Coffey, C.T. Kelley, and D.E. Keyes. Pseduo-Transient Continuation and Differential-Algebraic Equations. *SIAM Journal of Scientific Computing*, pages 1–13, 2002.

[8] Leonardo Dagum and Ramesh Menon. Openmp: An industry-standard api for shared-memory programming. *IEEE Comput. Sci. Eng.*, 5(1):46–55, January 1998.

[9] J. Dongarra, P. Beckman, T. Moore, P. Aerts, G. Aloisio, J.-C. Andre, D. Barkai, J.-Y. Berthou, T. Boku, B. Braunschweig, F. Cappello, B. Chapman, Xuebin Chi, a. Choudhary, S. Dosanjh, T. Dunning, S. Fiore, a. Geist, B. Gropp, R. Harrison, M. Hereld, M. Heroux, a. Hoisie, K. Hotta, Zhong Jin, Y. Ishikawa, F. Johnson, S. Kale, R. Kenway, D. Keyes, B. Kramer, J. Labarta, a. Lichnewsky, T. Lippert, B. Lucas, B. Maccabe, S. Matsuoka, P. Messina, P. Michielse, B. Mohr, M. S. Mueller, W. E. Nagel, H. Nakashima, M. E. Papka, D. Reed, M. Sato, E. Seidel, J. Shalf, D. Skinner, M. Snir, T. Sterling, R. Stevens, F. Streitz, B. Sugar, S. Sumimoto, W. Tang, J. Taylor, R. Thakur, a. Trefethen, M. Valero, a. van der Steen, J. Vetter, P. Williams, R. Wisniewski, and K. Yelick. The International Exascale Software Project roadmap. *International Journal of High Performance Computing Applications*, 25(1):3–60, 2011.

[10] David Eberly. Distance between point and triangle in 3D. *Magic Software...*, pages 1–6, 1999.

[11] Wolfgang Eckhardt. Efficient HPC Implementations for Large-Scale Molecular Simulation in Process Engineering. 2014.

[12] Wolfgang Eckhardt, Robert Glas, Denys Korzh, and Stefan Wallner. On-the-fly memory compression for multibody algorithms.

[13] Alexandre E. Eichenberger, Peng Wu, and Kevin O'Brien. Vectorization for SIMD architectures with alignment constraints. *ACM SIGPLAN Notices*, 39(6):82, 2004.

[14] C Ericson. The Gilbert-Johnson-Keerthi algorithm, 2005.

[15] Florian Fleissner and Peter Eberhard. Parallel Load Balanced Particle Simulation with Hierarchical Particle Grouping Strategies. pages 33–44.

[16] Message P Forum. Mpi: A message-passing interface standard. Technical report, Knoxville, TN, USA, 1994.

[17] Franz Franchetti and Markus Püschel. Generating SIMD vectorized permutations. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4959 LNCS:116–131, 2008.

[18] Robert M Freund. Algorithms for Constrained Optimization A . 1 Penalty and Barrier Methods. *Methods*, (23):55–58, 1968.

[19] Evghenii Gaburov and Yuri Cavecchi. XeonPhi Meets Astrophysical Fluid Dynamics. pages 1–7.

[20] L. Girolami, V. Hergault, G. Vinay, and A. Wachs. A three-dimensional discrete-grain model for the simulation of dam-break rectangular collapses: Comparison between numerical results and experiments. *Granular Matter*, 14(3):381–392, 2012.

[21] Pedro Gonnet. QuickSched: Task-based parallelism with dependencies and conflicts. pages 1–25, 2013.

[22] Pedro Gonnet. Efficient and Scalable Algorithms for Smoothed Particle Hydrodynamics on Hybrid Shared/Distributed-Memory Architectures. *SIAM Journal on Scientific Computing*, 37(1):C95–C121, 2015.

[23] P C Hansen. The L-Curve and its Use in the Numerical Treatment of Inverse Problems. *in Computational Inverse Problems in Electrocardiology, ed. P. Johnston, Advances in Computational Bioengineering*, 4:119–142, 2000.

[24] David Harmon, Etienne Vouga, Breannan Smith, Rasmus Tamstorf, and Eitan Grinspun. Asynchronous contact mechanics. *Communications of the ACM*, 55(3):102, 2012.

[25] David Harmon, Etienne Vouga, Rasmus Tamstorf, and Eitan Grinspun. Robust treatment of simultaneous collisions. *ACM Transactions on Graphics*, 27(3):1, 2008.

[26] Mark Hoemmen. Generating random numbers in parallel. pages 1–8, 2007.

[27] Kaixi Hou, Hao Wang, Wu-chun Feng, and Virginia Tech. ASPaS : A Framework for Automatic SIMDization of Parallel Sorting on x86-based Many-core Processors Categories and Subject Descriptors. pages 383–392.

[28] K. Iglberger and U. Rüde. Massively parallel granular flow simulations with non-spherical particles. *Computer Science - Research and Development*, 25(1-2):105–113, 2010.

[29] Klaus Iglberger and Ulrich Rüde. Massively parallel rigid body dynamics simulations. *Computer Science - Research and Development*, 23(3-4):159–167, 2009.

[30] Klaus Iglberger and Ulrich Rüde. Large-scale rigid body simulations. *Multibody System Dynamics*, 25(1):81–95, 2011.

[31] Adrian Jensen, Kirk Fraser, and George Laird. Improving the Precision of Discrete Element Simulations through Calibration Models. *13th International LS-DYNA Users Conference*, pages 1–12, 2014.

[32] Ben Juurlink. Performance Impact of Misaligned Accesses in SIMD Extensions. *Computer Engineering*, pages 334–342.

[33] Å Kaczmarczyk, Tomasz Koziara, and Cj Pearce. Corotational formulation for 3d solids. An analysis of geometrically nonlinear foam deformation. *arXiv preprint arXiv:1110.5321*, pages 1–23, 2011.

[34] Nils Karajan, Zhidong Han, Hailong Teng, and Jason Wang. On the Parameter Estimation for the Discrete-Element Method in LS-DYNA ®. *13 th International LS-DYNA Users Conference*, pages 1–9, 2014.

[35] Tero Karras. Maximizing Parallelism in the Construction of BVHs , Octrees , and k-d Trees. *EGGH-HPG'12 Proceedings of the Fourth ACM SIGGRAPH / Eurographics conference on High-Performance Graphics*, pages 33–37, 2012.

[36] Danny M. Kaufman, Timothy Edmunds, and Dinesh K. Pai. Fast frictional dynamics for rigid bodies. *ACM Transactions on Graphics*, 24(3):946, 2005.

[37] T. Koziara and N. Bicani. A distributed memory parallel multibody Contact Dynamics code. *International Journal for Numerical Methods in Engineering*, 87(1-5):437–456, 2011.

[38] Tomasz Koziara and Nenad Bićanić. Semismooth Newton method for frictional contact between pseudo-rigid bodies. *Computer Methods in Applied Mechanics and Engineering*, 197(33-40):2763–2777, 2008.

[39] Tomasz Koziara and Nenad Bićanić. Simple and efficient integration of rigid rotations suitable for constraint solvers. *International Journal for Numerical Methods in Engineering*, 81(9):1073–1092, 2010.

[40] Tomasz Koziara, Collision Detection, Sweep-plane Approach, Spatial Hashing, and Priority Search Tree. Sweep-plane approach to bounding box. *Complas VIII*, pages 1–4, 2005.

[41] Konstantinos Krestenitis and Tomasz Koziara. Calculating the Minimum Distance Between Two Triangles on SIMD Hardware. Number April, pages 8–11. 23th UK Conference on Computational Mechanics (ACME-UK). Swansea, UK., 2015.

[42] Konstantinos Krestenitis, Tobias Weinzierl, and Tomasz Koziara. A Multiscale DEM Contact Detection Code using Triangles for Non-Spherical Particle Simulations. Number April, pages 1–4. 24th UK Conference on Computational Mechanics (ACME-UK). Cardiff, UK., 2016.

[43] Shin-jae Lee, Minsoo Jeon, Dongseung Kim, and Andrew Sohn. Partitioned Parallel Radix Sort 1. *Journal of Parallel and Distributed Computing*, 668(October 2000):656–668, 2002.

[44] Xiang Ni, Laxmikant V. Kale, and Rasmus Tamstorf. Scalable Asynchronous Contact Mechanics Using Charm++. *2015 IEEE International Parallel and Distributed Processing Symposium*, pages 677–686, 2015.

[45] Igor P. Omelyan. Algorithm for numerical integration of the rigid-body equations of motion. *Physical Review E*, 58(1):1169–1172, 1998.

[46] Eric J R Parteli. DEM simulation of particles of complex shapes using the multisphere method: Application for additive manufacturing. *AIP Conference Proceedings*, 1542:185–188, 2013.

[47] Steve Plimpton. Fast Parallel Algorithms for Short – Range Molecular Dynamics. *Jounal of Computational Physics*, 117(June 1994):1–42, 1995.

[48] Chris H Rycroft, Terttaliisa Lind, Salih Güntay, and Abdel Dehbi. Granular flow in pebble bed reactors : dust generation and scaling. pages 447–455, 2012.

[49] Nadathur Satish, Mark Harris, and Michael Garland. Designing Efficient Sorting Algorithms for Manycore GPUs. *Proceedings of 23rd IEEE International Parallel and Distributed Processing Symposium*, pages 1–10, 2009.

[50] David E. Shaw. A fast, scalable method for the parallel evaluation of distance-limited pairwise particle interactions. *Journal of Computational Chemistry*, 26(13):1318–1328, 2005.

[51] Du Shen, Qi Luo, Denys Poshyvanyk, and Mark Grechanik. Automating performance bottleneck detection using search-based application profiling. *Proceedings of the 2015 International Symposium on Software Testing and Analysis - ISSTA 2015*, pages 270–281, 2015.

[52] Alice E Smith and David W Coit. Penalty functions. *Handbook of Evolutionary Computation*, 97(1):C5, 1995.

[53] Jerome M. Solberg and Panayiotis Papadopoulos. Impact of an elastic pseudo-rigid body on a rigid foundation. *International Journal of Engineering Science*, 38(6):589–603, 2000.

[54] Xinmin Tian, Hideki Saito, Serguei V. Preis, Eric N. Garcia, Sergey S. Kozhukhov, Matt Masten, Aleksei G. Cherkasov, and Nikolay Panchenko. Effective SIMD Vectorization for Intel Xeon Phi Coprocessors. *Scientific Programming*, 2015, 2015.

[55] J. Treibig, G. Hager, and G. Wellein. Likwid: A lightweight performance-oriented tool suite for x86 multicore environments. In *Proceedings of PSTI2010, the First International Workshop on Parallel Software Tools and Tool Infrastructures*, San Diego CA, 2010.

[56] Oren Tropp, Ayellet Tal, and Ilan Shimshoni. A fast triangle to triangle intersection test for collision detection. *Computer Animation and Virtual Worlds*, 17(5):527–535, 2006.

[57] Anthony Wachs, Laurence Girolami, Guillaume Vinay, and Gilles Ferrer. Grains3D, a flexible DEM approach for particles of arbitrary convex shape - Part I: Numerical model and validations. *Powder Technology*, 224:374–389, 2012.

[58] Anthony Wachs and Andriarimina Daniel Rakotonirina. A MPI / domain decomposition strategy for large-scale simulations of granular media made of particles of arbitrary shape. 130(2011):69360, 2012.

[59] J R Williams and R O Connor. Discrete Element Simulation and the Contact Problem. *Methods in Engineering*, 6(June 1996):279–304, 1999.

[60] Afra Zomorodian and Herbert Edelsbrunner. Fast Software for Box Intersections. *International Journal of Computational Geometry & Applications*, 12(01n02):143–172, 2002.