Comparison Between $O(n^2)$ and $O(n)$ Neighbor Search Algorithm and its Influence on Superlinear Speedup in Parallel DEM for Complex-shaped Particles

Beichuan Yan[a,*], Richard A. Regueiro[a]

[a]*Department of Civil, Environmental, and Architectural Engineering, University of Colorado Boulder*

Abstract

This paper presents performance comparison between $O(n^2)$ and $O(n)$ neighbor search algorithms, studies their effects for different particle shape complexity and computational granularity (CG), and investigates the influence on superlinear speedup of 3D Discrete Element Method (DEM) for complex-shaped particles. In serial computing, the more complex the particle shapes (from sphere to ellipsoid to poly-ellipsoid), the smaller the neighbor search fraction (NSF); and the lower the CG, the smaller the NSF. The $O(n)$ search algorithm is superior to $O(n^2)$ on the whole, especially for situations with coarser CG; however, the $O(n)$ algorithm is inferior to $O(n^2)$ for CG finer than the parallel optimal CG.

In parallel computing of complex-shaped particles, $O(n^2)$ algorithm is slower than $O(n)$ at very coarse CGs; however, it becomes faster than $O(n)$ at fine CGs that are mostly employed in practical computations to achieve best performance. This means that $O(n^2)$ algorithm outperforms $O(n)$ algorithm in large-scale parallel 3D DEM simulations generally.

Both $O(n^2)$ and $O(n)$ algorithms exhibit a strong superlinear speedup for large scale simulations of complex-shaped 3D DEM. The $O(n)$ algorithm always exhibits a lower speedup than the $O(n^2)$ algorithm across all computational scales and granularities, mostly due to the fact that the wall time ratio between $O(n^2)$ and $O(n)$ algorithms increases with an increase of CG,

*Corresponding author. Fax: +1-303-492-7317

Email addresses: beichuan.yan@colorado.edu (Beichuan Yan), richard.regueiro@colorado.edu (Richard A. Regueiro)

until only one compute node is used. On average, the speedup in $O(n)$ algorithm is reduced by approximately 1/3 relative to $O(n^2)$ algorithm on the simulation scale of 1 million ellipsoidal particles. In addition, the weak scaling measurements reveal close-to-linear scalability in terms of nearly constant computation time.

Key words: discrete element, complex-shaped particles, neighbor search, contact resolution, superlinear speedup, computational granularity

---

1. MOTIVATION

In DEM simulations, there are three typical neighbor search algorithms with different time complexities: $O(n^2)$, coming from n-by-n simple search; $O(n\log n)$, resulting from *tree-based algorithms* [Jagadish et al., 2005, Muja and Lowe, 2009]; and $O(n)$, rooted from *binning algorithms* [Munjiza and Andrews, 1998, Williams et al., 2004] or link-cell (LC) method [Grest et al., 1989], where n denotes the number of particles. How these different algorithms influence DEM computational performance, not only for serial computing, but especially for parallel computing, remains unanswered. In particular, what difference they would make with change of particle shape (from sphere to ellipsoid to poly-ellipsoid, and even more complex shapes) and change of computational granularity (namely, number of particles per CPU core) still needs investigation. Furthermore, how these algorithms would affect the superlinear speedup phenomenon [Yan and Regueiro, 2018c] observed in 3D DEM parallel computing of complex-shaped particles is of great interest when evaluating computational speedup and efficiency. In addition, many times it has been taken for granted, for example, that $O(n)$ algorithm always performs better than $O(n^2)$; however, it turns out not to be the case.

This paper presents comparison between $O(n^2)$ and $O(n)$ neighbor search algorithms, studies their effects for particle shape complexity and computational granularity (CG), and investigates the influence on superlinear speedup of 3D DEM of complex-shaped particles. It contains eight sections: section 1 has stated the motivation; section 2 provides an introduction to the computational features and structure of DEM, and defines several fundamental concepts; section 3 describes the implementation of $O(n^2)$ and $O(n)$ neighbor search algorithms in both serial and parallel computing; section 4 analyzes $O(n^2)$ and $O(n)$ performance in serial computing and the influence of particle shapes and CG; section 5 compares the performance of $O(n^2)$ and $O(n)$ algo-

rithms in parallel computing across 4 to 5 orders of magnitude of simulation scale for complex-shaped 3D DEM on Department of Defense (DoD) supercomputers; section 6 focuses on the superlinear speedup of complex-shaped 3D DEM and evaluates the influence of $O(n^2)$ and $O(n)$ algorithms on it; section 7 demonstrates the close-to-linear scalability in terms of weak scaling measurements; and a summary is given in the last section.

2. FUNDAMENTALS OF DEM

2.1. The DEM framework

A complete DEM system is composed of multiple essential components: particle geometry representation, interparticle mechanical models (such as Hertz nonlinear normal contact model [Hertz, 1882] and Mindlin's history-dependent shear model [Mindlin, 1949, Mindlin and Deresiewicz, 1953]), contact search and resolution algorithm, time integration scheme, damping mechanism, boundary control methods for modeling various loading conditions, etc. The DEM is computationally expensive owing to the following factors: non-linear and history-dependent interparticle contact mechanical models, complicated contact geometry resolution between particle pairs, and small time step induced by explicit time integration scheme. A typical procedure of DEM analysis consists of three major computational steps in sequence, which are integrated in time using central difference method until a simulation is completed:

1. contact detection between particles, including two phases:
   (a) *neighbor search (neighbor estimate)*
   (b) *contact resolution*
2. contact force computation for each pair of particles in contact.
3. particle motion update (translations and rotations) using Newton's second law.

The contact detection process is usually the major computational bottleneck, especially for a large number of complex-shaped particles. It is divided into two phases: (a) the neighbor search (or spatial reasoning) phase, and (b) the contact resolution phase. Neighbor search identifies/estimates objects near the target object. It often uses an approximate geometry for the objects, such as bounding box or bounding sphere. In the paper we use the

phrases neighbor search and neighbor estimate interchangeably. The geometric contact resolution phase then uses a specific geometric representation of each body to resolve the contact geometry. For complex shapes such as ellipsoidal particles (three different semi-axis lengths) [Yan et al., 2010] or non-symmetric poly-ellipsoidal particles [Peters et al., 2009], the contact resolution between two particles is much more expensive than spheres, increasing the floating point operations by several orders of magnitude due to the requirement of numerical accuracy and robustness. This is the most computationally challenging part of 3D DEM in addition to the non-linear and history-dependent mechanical models that describe interparticle interactions.

2.2. Neighbor search

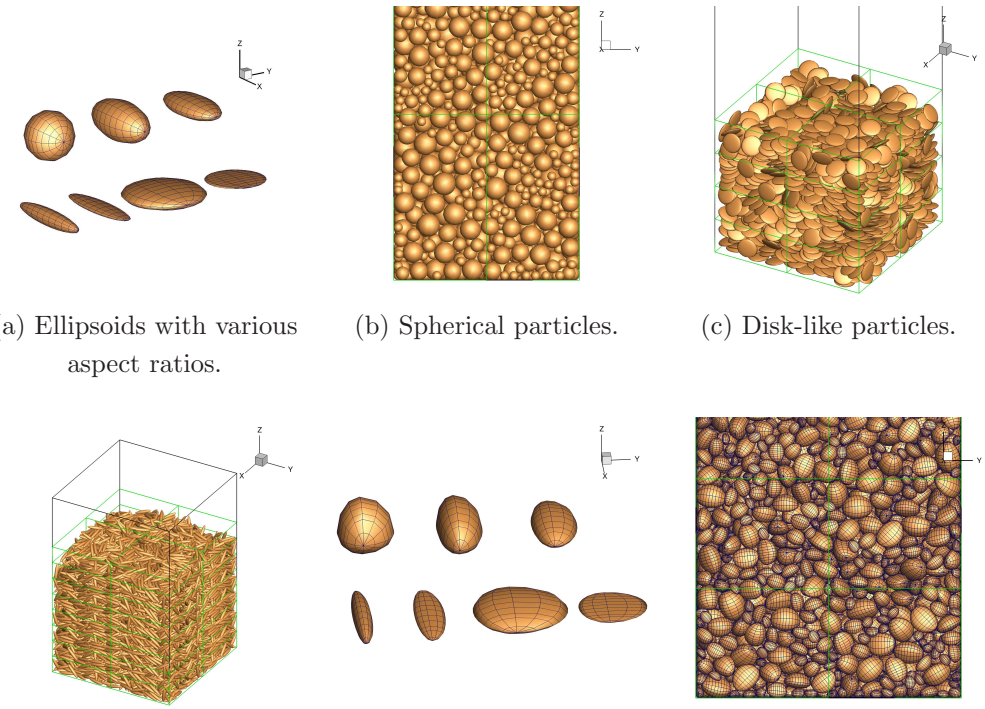
As mentioned in Section 1, the neighbor search algorithm can have different time complexities: $O(n^2)$, $O(n\log n)$ and $O(n)$. The $O(n)$ algorithm is achieved either from *no binary search (NBS)* [Munjiza and Andrews, 1998], *binning algorithms* [Williams et al., 2004] or *link-cell (LC) method* [Grest et al., 1989].

The three methods are essentially the same, and they are just different names used in geomechanics and molecular dynamics (MD). For example, the idea of binning algorithm is to place each particle into a bin using a hash on the particle's coordinates. Once the particles are sorted into bins, one can reason about the spatial closeness based solely on the fixed relationships of the bins. Munjiza and Andrews [1998] implemented the NBS, a binning algorithm which scales linearly to large numbers of particles but is limited to particles of approximately the same size. Williams et al. [2004] extended the traditional binning algorithm so that objects of arbitrary shape and size in two and three dimensions can be handled by introducing an abstraction. The algorithm achieves the partitioning of n particles of arbitrary shape and size into n lists in $O(n)$ operations, where each list consists of particles spatially near to the target object; the LC method divides a computational spatial domain into equal cubical cells of size not smaller than the cutoff distance (in MD) or diameter of the largest particle (in DEM). Each particle is referenced to the cell according to the position of the particle centroid. The neighbor estimate comprises referencing of individual particles to the cells and constructing of the neighbors list of particles using surrounding cells.

It is worth noting that the three algorithms, $O(n^2)$, $O(n\log n)$ and $O(n)$, only affect the performance of neighbor estimate. They have no bearing on

contact resolution. And it can be imagined that the overall performance improvement resulting from these algorithms might be highly limited for complex-shaped particles, because neighbor estimate only takes up a small fraction of floating point operations in the whole computation.

2.3. Contact resolution



(a) Ellipsoids with various aspect ratios.

(b) Spherical particles.

(c) Disk-like particles.

(d) Needle-like particles.

(e) Poly-ellipsoids with various aspect ratios.

(f) Poly-ellipsoids.

Figure 1: Ellipsoids and poly-ellipsoids represent a wide variety of shapes in DEM.

Yan et al. [2010] developed a robust contact resolution algorithm for true ellipsoidal particles by constructing an *extreme value problem* of finding the deepest penetration of one particle into the other. Such an extreme value problem results in a sixth order polynomial equation. Conventional polynomial root finders cannot satisfy the high-precision numerical requirement in the 3D DEM computation. For example, the elastic overlap between two particles of typical quartz sand may vary between 10^{-8} to 10^{-5} meters depending on particle size, shape and external loading, and a low-precision solver can

lead to numerical instability or spurious explosion of particles. Therefore, an iterative eigenvalue method is selected to find roots of the polynomial and determine the contact geometry.

The algorithm and its implementation has been shown to be robust such that it is applicable to not only regularly bulky ellipsoidal shapes but also extreme-shaped ellipsoidal particles such as disks and needles, as shown in Figure 1(a∼d).

Peters et al. [2009] proposed a non-symmetric poly-ellipsoid shape which joins eight component ellipsoids in eight different octants respectively to produce continuous surface coordinates, normal directions and intersections. It is more computationally expensive than a symmetric ellipsoid but it acts as a useful extension, shown in Figure 1(e∼f).

It is worth emphasizing that the algorithm comparison pertains to complex-shaped particles such as ellipsoids [Yan et al., 2010], poly-ellipsoids [Peters et al., 2009], superellipsoids [Wellmann et al., 2008, Delaney et al., 2010], superquadrics [Williams and Pentland, 1992] or asymmetrical particles constructed by non-uniform rational Basis-Splines (NURBS) [Lim and Andrade, 2014], rather than simplistic spheres; we specifically focus on ellipsoids and poly-ellipsoids in the paper. In many natural phenomena and engineering problems, the shapes (and sizes, gradations, etc) of the discrete particles play an insurmountably important role such as for capturing particle interlocking and particle fracture.

3. $O(n^2)$ and $O(n)$ IMPLEMENTATIONS

3.1. Implementations in serial computing

Figure 2 illustrates the $O(n^2)$ and $O(n)$ algorithms in serial computing. In $O(n^2)$ algorithm each particle is checked against all other particles to identify potentially interactive particle pairs, therefore space division is not needed. Within one step the particle pairs are pushed into data structures such as vector or linked-list for subsequent contact resolution. In $O(n)$ algorithm two sequential steps have to be followed: (1) particle assignment, which subdivides the computational domain into equally-sized cuboidal cells (white cells shown in Figure 2) and each particle must be assigned to a cell in terms of its centroid location; and (2) neighbor search, which not only searches contact pairs inside each cell, but also finds contact pairs between a cell and its surrounding cells.

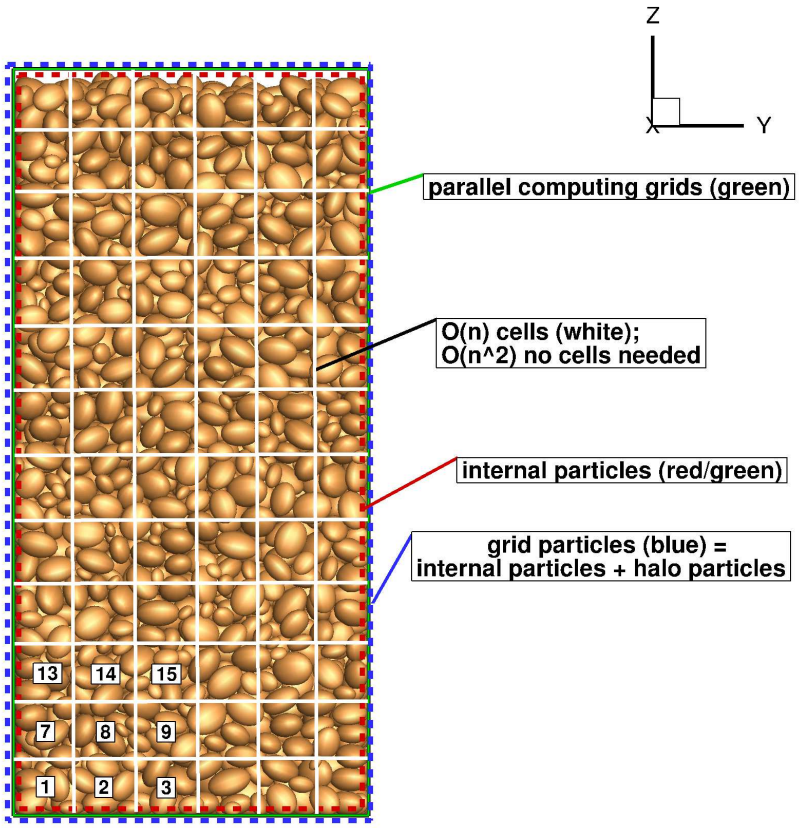


Figure 2: $O(n^2)$ and $O(n)$ search algorithms in serial computing: 1 process handles all particles.

The particle assignment can work in $O(n)$ rather than $O(n^2)$ time complexity, based on the assumption that cuboidal cells are equally-sized in either of the three dimensions independently. For instance, to identify in which cell a particle should be located along x direction, the following equation can be applied,

$$i_x = Integer\left(\frac{x - x_{min}}{\Delta d_x}\right) \tag{1}$$

where x is the centroid of the particle, x_{min} is the minimum coordinates in x direction, Δd_x denotes the cell dimension in x direction, $Integer$ denotes type conversion operator, and i_x is the desired cell coordinate in x direction.

Referring to Figure 2, a serial $O(n)$ algorithm marks each cell that has been searched in sequence, and guarantees that the cell will not be searched again to generate duplicate contact pairs when cycling through all of the cells. For example, cell 1 checks with cells 7, 8 and 2; cell 2 only checks with cells 7, 8, 9, 3; and cell 8 only checks with cells 13, 14, 15 and 9, when the search occurs in sequence in space. Since it is serial computing, there are actually no halo particles received from other processes.

It is easily imagined that the $O(n)$ algorithm might be inferior to $O(n^2)$ for computing a very small number of particles, due to its particle assignment overhead, which is verified in Section 4.

3.2. Parallel computing framework

3.2.1. Link-block and four-step MPI design

We upscale the link-cell (LC) to link-block (LB) technique in parallel DEM, and apply Foster's four-step paradigm in designing a parallel algorithm in DEM.

Partitioning: The computational domain is divided into blocks. Each block may consist of many virtual cells. In Figure 3, there are 8 blocks numbered from 0 to 7, each containing 5 x 5 x 5 small virtual cells. The size of the virtual cells is chosen to be the maximum diameter of the discrete particles.

Communication: Each cell, as a primitive task unit, can communicate with 26 possible surrounding ones to determine contact detection. However, the communication manner may be changed after the process of agglomeration.

Agglomeration: By combining 5 x 5 x 5 virtual cells into a block, communication overhead is lowered in that each block only needs to communicate

with neighboring blocks through border/ghost layers, which are virtual cells marked by blue dots in Figure 3.

Mapping: There are choices of mapping a block of particles to a core, a CPU, multiCPUs within a node, or even a whole node. Very often each block is mapped to a whole compute node.

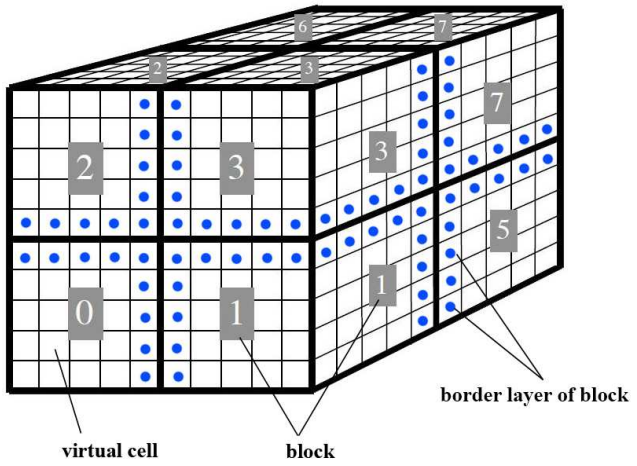


Figure 3: Link-blocks, virtual cells and border layers.

3.2.2. Flowchart of the parallel algorithm

The flowchart of the parallel algorithm is depicted in Figure 4. For brevity, the serial algorithm is not illustrated separately, because it only includes the following steps shown in Figure 4: steps 1, 4, 5, 6, 7 and 8. In comparison to the serial code (Ellip3d), the parallel code (ParaEllip3d) ends up with six more steps as follows:

1. step 2: 2-Root process divides and broadcasts info. This step only runs once so it does not cost much CPU time.
2. step 3: 3-All processes communicate with neighbors. This interprocess communication is the most important step in the parallel algorithm.
3. step 9: 9-All processes update compute grids. This step arises from consideration of computational load balance.
4. step 10: 10-All processes merge and output info. This step serves the goal of snapshotting simulation states. Beware that it does not execute at each time increment, otherwise it could cause unacceptable cost.

5. step 11: 11-All processes release memory of receiving particle info. This step arises from MPI transmission mechanism and must be carefully taken care of.
6. step 12: 12-All processes migrate particles. This step handles the situation when particles move across block borders.

3.3. Implementations in parallel computing

Figure 5 illustrates the $O(n^2)$ and $O(n)$ algorithms in parallel computing, whereby 2x2x3 compute grids/blocks in x, y, z direction respectively (in green) are used as an example. A block (or compute grid) not only manages its internal particles (enclosed by green or red dashed lines), but also handles halo particles (enclosed between the red and blue dashed lines) transmitted from neighboring blocks through MPI.

The $O(n^2)$ algorithm remains the same as long as it processes both internal particles and halo particles as a whole. The $O(n)$ algorithm needs to extend the equally-sized cells from red line enclosed volume to blue line enclosed volume, which is required in order to keep track of all particles accurately in numerical computations.

4. $O(n^2)$ VS $O(n)$ IN SERIAL COMPUTING

A Dell T7500 Precision Linux workstation of dual hexa-core Intel Xeon X5690 processors is used to carry out performance tests for serial computing. It is shown that the complex-shaped particles are much more computationally demanding than simple spheres. For instance, the contact resolution between a pair of true ellipsoids is approximately 50 times as expensive as that of a pair of spheres, and contact resolution between a pair of poly-ellipsoids is nearly 300 times as expensive as that of a pair of spheres.

Serial computing tests on the $O(n^2)$ and $O(n)$ algorithms are carried out for different particle shapes (sphere, ellipsoid and poly-ellipsoid) and different computational granularity (CG) of 50, 150, 500, 2k and 5k particles. The neighbor search fraction (NSF) is defined as follows:

$$NSF = \frac{T_{\text{neighbor estimate}}}{T_{\text{contact detection time}}} = \frac{T_{\text{neighbor estimate}}}{T_{\text{neighbor estimate}} + T_{\text{contact resolution}}},$$

where T denotes the wall clock time.

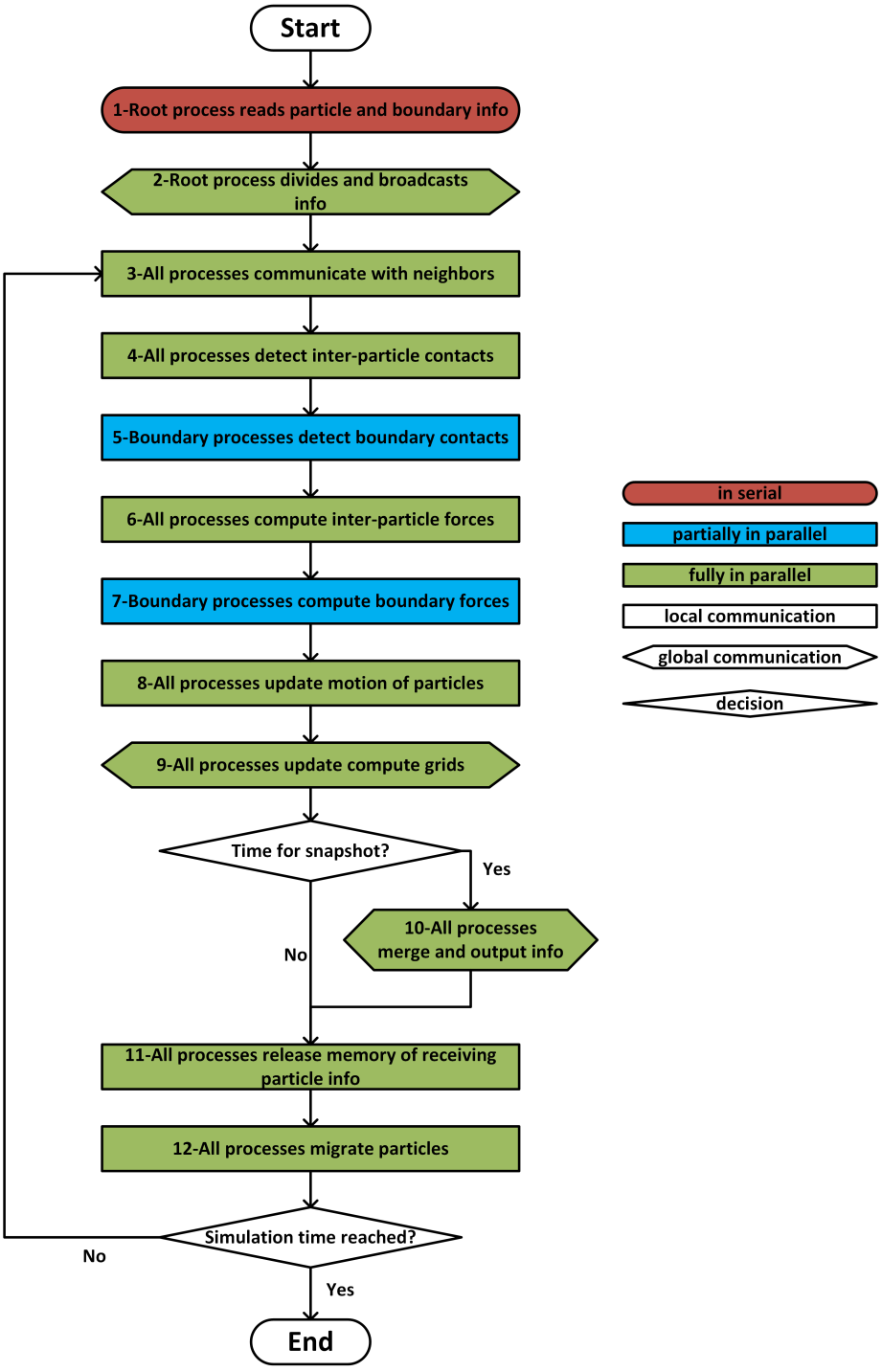


Figure 4: Flowchart of the parallel algorithm of 3D DEM.

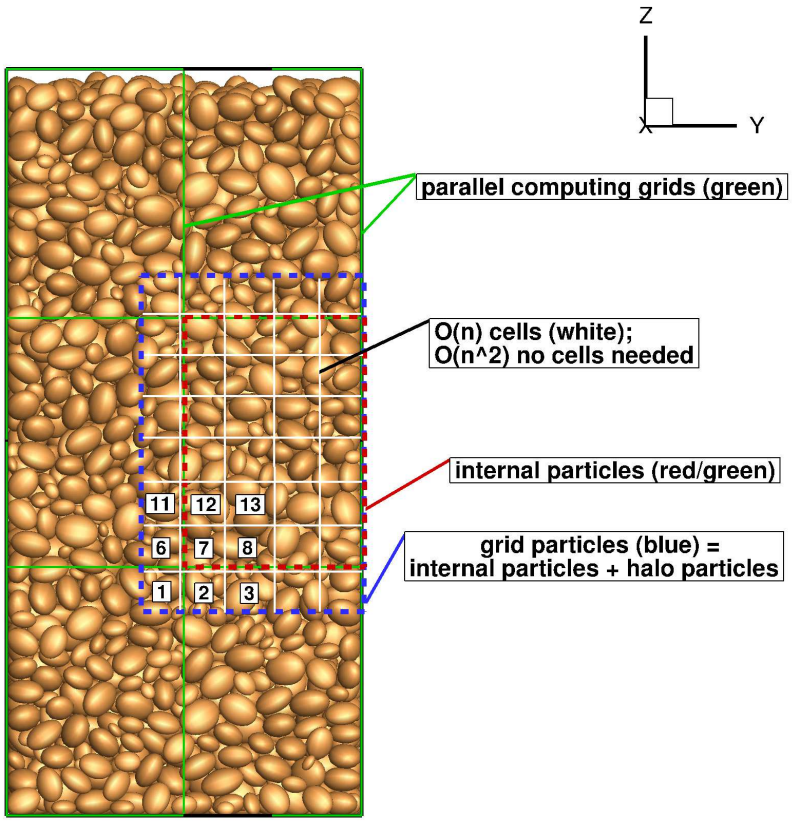
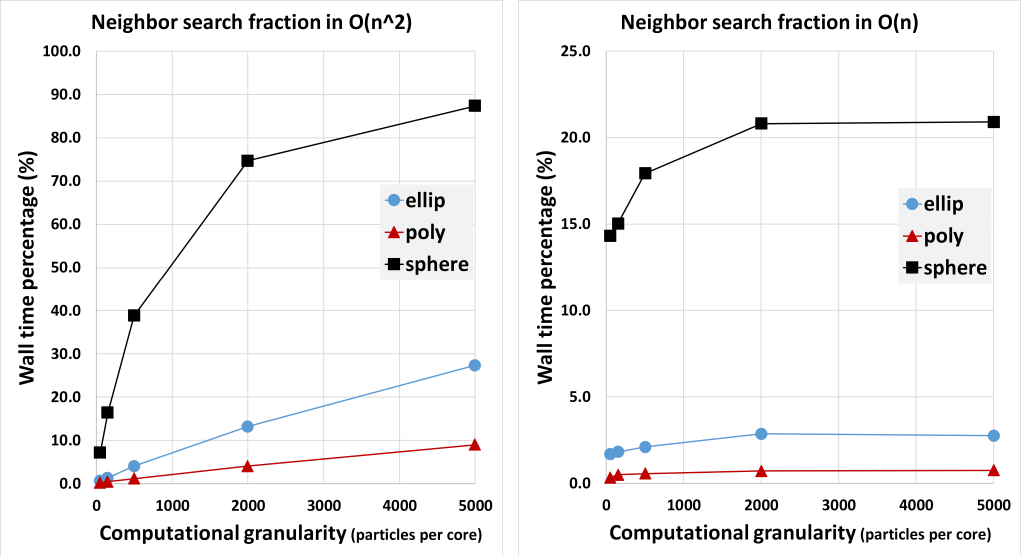


Figure 5: $O(n^2)$ and $O(n)$ search algorithms in parallel computing, e.g., 2x2x3 processes in x, y, z direction, respectively.



(a) $O(n^2)$. (b) $O(n)$.

Figure 6: $O(n^2)$ and $O(n)$ algorithms on three particle shapes.

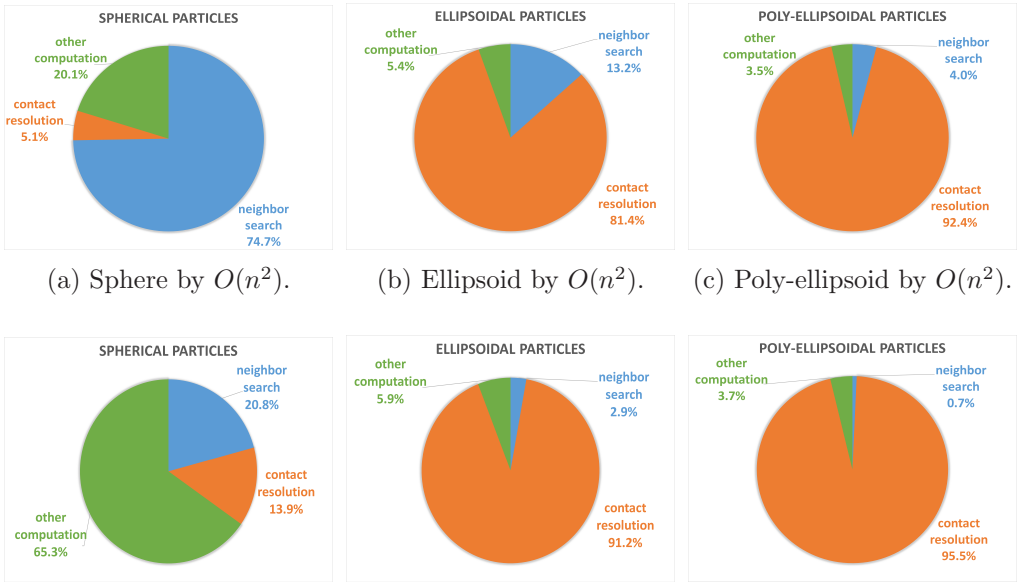
Figure 6 presents the NSF of $O(n^2)$ and $O(n)$ algorithms for three particle shapes and 5 CGs. It is observed that the more complex the shapes are (from sphere to ellipsoid to poly-ellipsoid), the smaller the NSF is; and the lower the CG is, the smaller the NSF is. The data for ellipsoid and poly-ellipsoid are listed in Table 1. As a typical example, the NSFs are 1.3% for $O(n^2)$ and 1.8% for $O(n)$ on ellipsoid, and 0.4% for $O(n^2)$ and 0.5% for $O(n)$ on poly-ellipsoid at the parallel optimal CG (150 $\sim$ 300 particles per core (PPC)).

Table 1: NSF(%) of Ellipsoid and Poly-ellipsoid

	Ellipsoid		Poly-Ellipsoid	
# of particles	$O(n^2)$	$O(n)$	$O(n^2)$	$O(n)$
50	0.6	1.7	0.2	0.3
150	1.3	1.8	0.4	0.5
500	4.0	2.1	1.1	0.6
2000	13.2	2.9	4.0	0.7
5000	27.3	2.8	9.0	0.8

Beware the parallel optimal CGs are obtained by testing various number

(from one to excessive) of compute nodes and selecting the one which achieves shortest wall clock time from parallel computing. Yan and Regueiro [2018b] pointed out: for a fixed problem size (or simulation scale), the parallel overhead (mostly the interprocess communication) percentage increases with a decreasing CG, due to the fact that computation time decreases faster than the communication time when the CG decreases in 3D DEM of complex-shaped particles, and the best performance is achieved at a certain CG in parallel computing. It should be emphasized that the the parallel optimal CG of 150 $\sim$ 300 particles per core (PPC) is measured for complex-shaped particles; for spheres it is much larger due to computational cost difference, e.g., 15, 000 $\sim$ 30, 000 PPC.



(a) Sphere by $O(n^2)$. (b) Ellipsoid by $O(n^2)$. (c) Poly-ellipsoid by $O(n^2)$.

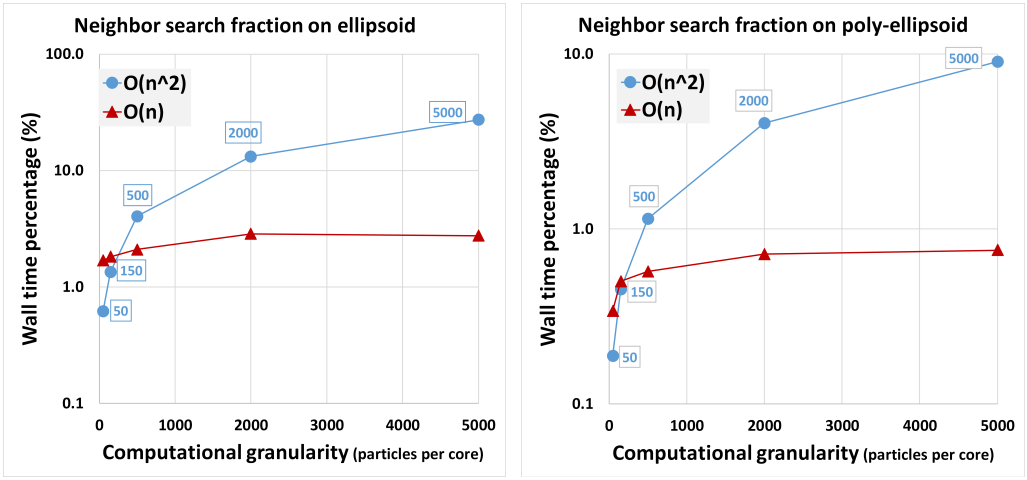
(d) Sphere by $O(n)$. (e) Ellipsoid by $O(n)$. (f) Poly-ellipsoid by $O(n)$.

Figure 7: Wall clock time percentage of DEM components by $O(n^2)$ and $O(n)$ algorithms using different particle shapes.

Figure 7 plots pie charts on time percentage of DEM components using 2,000 particles with different shapes, and it confirms that the more complex the particle shapes are, the smaller the NSF is. With poly-ellipsoid computing by $O(n)$ algorithm, the NSF is as low as 0.7% whereas contact resolution takes up to 95.5%.

Figure 8 compares the NSFs of $O(n^2)$ and $O(n)$ algorithms on ellipsoid and poly-ellipsoid. It can be seen that the NSF in $O(n^2)$ algorithm exhibits

an overall higher percentage range, up to 27% for ellipsoid and 9% for poly-ellipsoid, than that of the O(n) algorithm, up to 2.8% for ellipsoid and 0.8% for poly-ellipsoid, across 5 computational granularity (CG) of 50, 150, 500, 2k and 5k, respectively.



(a) Ellipsoid. (b) Poly-ellipsoid.

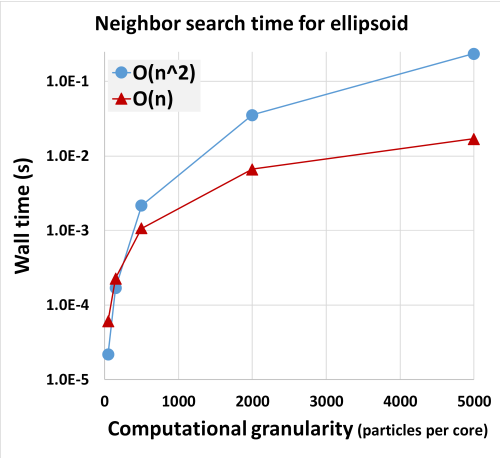
Figure 8: NSFs of $O(n^2)$ and $O(n)$ algorithms on ellipsoid and poly-ellipsoid.

Figure 9 compares the neighbor search time, and Figure 10 compares the contact detection time, of $O(n^2)$ and $O(n)$ algorithms on ellipsoid and poly-ellipsoid, respectively.

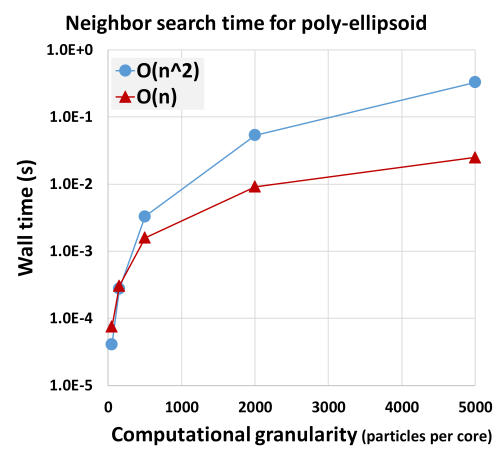
It is clear that the NSF, neighbor search time and contact detection time increase faster with regard to CG in $O(n^2)$ algorithm than it does in $O(n)$ algorithm, and the gap between $O(n^2)$ and $O(n)$ algorithms increases with the increase of CG. That indicates the $O(n)$ search algorithm is superior to $O(n^2)$ on the whole in serial computing, especially for situations with coarser CG.

However, it is worth noting that the $O(n^2)$ and $O(n)$ algorithms perform nearly the same at the parallel optimal CG ($150 \sim 300$ PPC), which is observed for NSF (Figure 8), neighbor search time (Figure 9) and contact detection time (Figure 10). In particular, the NSF and neighbor search time of $O(n^2)$ algorithm are even lower than that of $O(n)$ algorithm when the CG goes finer than 150 PPC. In other words, the $O(n)$ search algorithm is inferior to $O(n^2)$ for CG finer than the parallel optimal one.

Table 2 lists the overall serial computing time, and it is seen that the

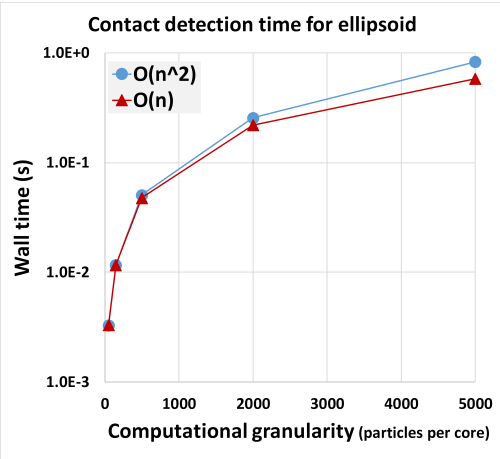


(a) Ellipsoid.

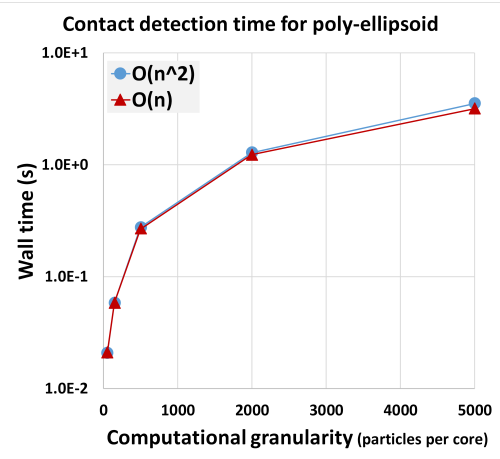


(b) Poly-ellipsoid.

Figure 9: Neighbor search time of $O(n^2)$ and $O(n)$ algorithms on ellipsoid and poly-ellipsoid.



(a) Ellipsoid.



(b) Poly-ellipsoid.

Figure 10: Contact detection time of $O(n^2)$ and $O(n)$ algorithms on ellipsoid and poly-ellipsoid.

Table 2: Overall serial computing wall clock time (s)

# of particles	Ellipsoid $O(n^2)$	Ellipsoid $O(n)$	ratio	Poly-Ellipsoid $O(n^2)$	Poly-Ellipsoid $O(n)$	ratio
50	3.56E-03	3.61E-03	0.984	2.19E-02	2.21E-02	0.989
150	1.26E-02	1.25E-02	1.002	6.15E-02	6.14E-02	1.002
500	5.40E-02	5.12E-02	1.055	2.89E-01	2.79E-01	1.037
2000	2.71E-01	2.35E-01	1.155	1.33E+00	1.28E+00	1.043
5000	8.66E-01	6.18E-01	1.401	3.66E+00	3.31E+00	1.108

time ratio of $O(n^2)$ to $O(n)$ is 1.002 at approximately parallel optimal CG, and it even goes below 1 for finer CG than the parallel optimal one.

Tables 3, 4, 5 list the time of neighbor estimate, contact resolution and overall computation, respectively. The average time ratios of poly-ellipsoid to ellipsoid are 1.5, 5.5, and 5.1 in neighbor estimate, contact resolution and overall computation, respectively.

Table 3: Neighbor estimate time

wall clock time (s)	$O(n^2)$				$O(n)$			
	150	500	2000	5000	150	500	2000	5000
sphere	1.44E-04	1.80E-03	3.71E-02	2.26E-01	1.28E-04	6.26E-04	3.09E-03	8.30E-03
ellip	1.69E-04	2.19E-03	3.57E-02	2.37E-01	2.28E-04	1.08E-03	6.70E-03	1.70E-02
poly	2.76E-04	3.31E-03	5.36E-02	3.31E-01	3.07E-04	1.59E-03	9.16E-03	2.49E-02
poly/ellip	1.6	1.5	1.5	1.4	1.3	1.5	1.4	1.5

Table 4: Contact resolution time

wall clock time (s)	$O(n^2)$				$O(n)$			
	150	500	2000	5000	150	500	2000	5000
sphere	1.18E-04	4.72E-04	2.56E-03	7.41E-03	1.20E-04	4.81E-04	2.06E-03	5.64E-03
ellip	1.15E-02	4.86E-02	2.21E-01	5.94E-01	1.14E-02	4.69E-02	2.14E-01	5.66E-01
poly	5.81E-02	2.74E-01	1.23E+00	3.21E+00	5.79E-02	2.66E-01	1.22E+00	3.16E+00
poly/ellip	5.0	5.6	5.6	5.4	5.1	5.7	5.7	5.6

5. $O(n^2)$ VS $O(n)$ IN PARALLEL COMPUTING

The parallel simulations are performed on a DoD HPC supercomputer, Spirit, which is an SGI ICE X System located at the AFRL DSRC. Spirit has 4,590 compute nodes each with 16 cores (73,440 total compute cores), 146.88 TBytes of memory, and is rated at 1.5 peak PFLOPS. Each compute node

Table 5: Overall computation time

wall clock time (s)	$O(n^2)$				$O(n)$			
	150	500	2000	5000	150	500	2000	5000
sphere	8.78E-04	4.62E-03	4.97E-02	2.59E-01	8.52E-04	3.49E-03	1.48E-02	3.97E-02
ellip	1.26E-02	5.40E-02	2.71E-01	8.66E-01	1.25E-02	5.12E-02	2.35E-01	6.18E-01
poly	6.15E-02	2.89E-01	1.33E+00	3.66E+00	6.14E-02	2.79E-01	1.28E+00	3.31E+00
poly/ellip	4.9	5.4	4.9	4.2	4.9	5.4	5.5	5.3

has two Sandy Bridge-based Intel Xeon CPU E5-2670 2.60GHz and 32 GB memory. The cluster of compute nodes are interconnected through FDR 14x InfiniBand network with enhanced LX hypercube topology. A combination of the following compilers and libraries is used: Intel compilers 16.0.3, SGI MPT 2.14 and Boost C++ 1.57.

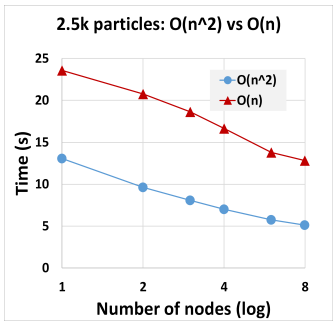
Figure 11 plots the wall clock time consumed across 5 orders of magnitude of simulation scale, 2.5k, 12k, 150k, 1M and 10M ellipsoidal particles, each of which tests various number of compute nodes. In other words, various CGs are tested for each scale.

On the scales of 2.5k and 12k particles plotted in Figure 11(a, b), $O(n^2)$ algorithm is always faster than $O(n)$, across 1 to 64 compute nodes. Note the CGs are fairly fine in these two cases; for example, the CG is 156 particles per core (PPC) using 1 node and 39 PPC using 4 nodes for 2.5k particles, and the CG is 750 PPC using 1 node and 47 PPC using 16 nodes for 12k particles.

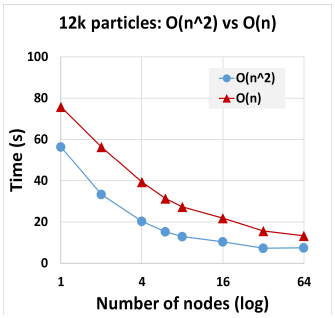
On the scales of 150k and 1M particles plotted in Figure 11(c, d), $O(n^2)$ algorithm is slower than $O(n)$ with a few compute nodes, however it becomes faster than $O(n)$ with appropriate number of compute nodes used in practice. For example, $O(n^2)$ algorithm is slower than $O(n)$ using 1 node (9,375 PPC) to 2 nodes (4,688 PPC) but faster using 4 nodes (2,344 PPC) to 128 nodes (73 PPC) for 150k particles, and $O(n^2)$ algorithm is slower than $O(n)$ using 1 node (62,500 PPC) to 32 nodes (1,953 PPC) but faster using 64 nodes (977 PPC) to 512 nodes (122 PPC) for 1M particles. These are clearly plotted with log-log graphs in Figure 11(c', d').

Figure 12 plots the wall clock time consumed across 4 orders of magnitude of simulation scale, 2.5k, 12k, 150k and 1M poly-ellipsoidal particles, each of which tests various number of compute nodes.

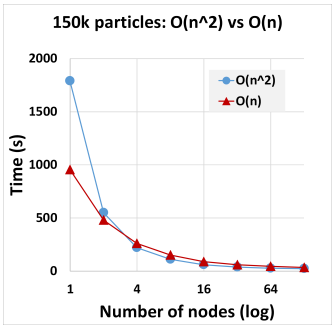
On the scales of 2.5k and 12k particles plotted in Figure 12(a, b), $O(n^2)$ algorithm is always faster than $O(n)$, across 1 to 64 compute nodes. Note like the ellipsoid cases, the CGs are fine in these two cases, for example, the CG is 156 particles per core (PPC) using 1 node and 39 PPC using 4 nodes



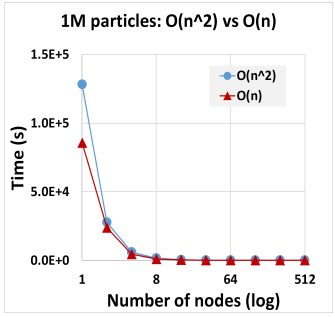
(a) 2.5k particles (log).



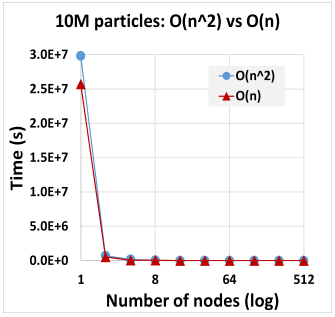
(b) 12k particles (log).



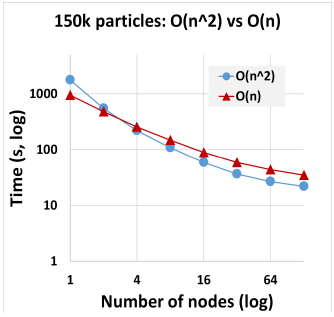
(c) 150k particles (log).



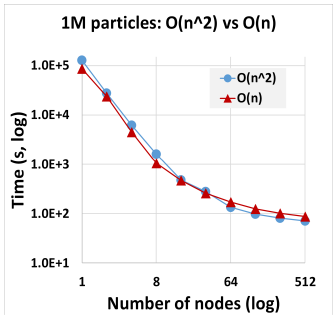
(d) 1M particles (log).



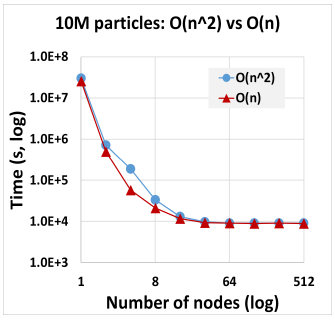
(e) 10M particles (log).



(c') 150k particles (log-log).

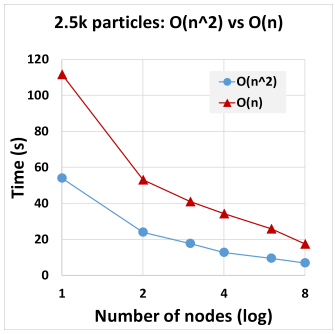


(d') 1M particles (log-log).

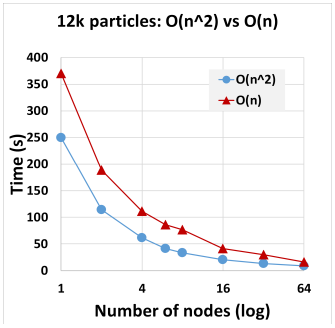


(e') 10M particles (log-log).

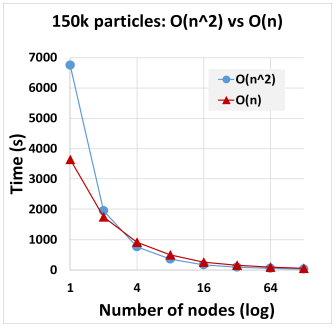
Figure 11: Wall clock time in computing ellipsoids across 5 orders of magnitude of simulation scale.



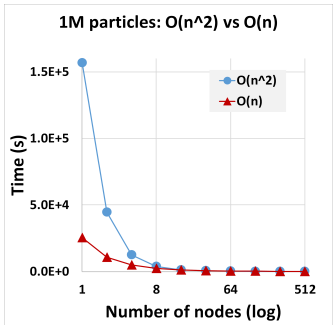
(a) 2.5k particles (log).



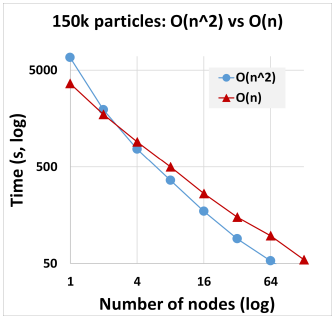
(b) 12k particles (log).



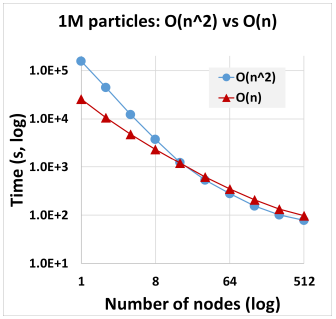
(c) 150k particles (log).



(d) 1M particles (log).



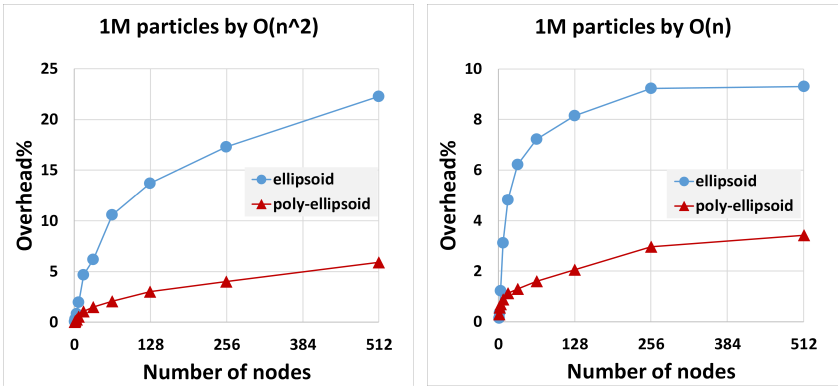
(c') 150k particles (log-log).



(d') 1M particles (log-log).

Figure 12: Wall clock time in computing poly-ellipsoids across 4 orders of magnitude of simulation scale.

for 2.5k particles, and the CG is 750 PPC using 1 node and 47 PPC using 16 nodes for 12k particles.



(a) $O(n^2)$. (b) $O(n)$.

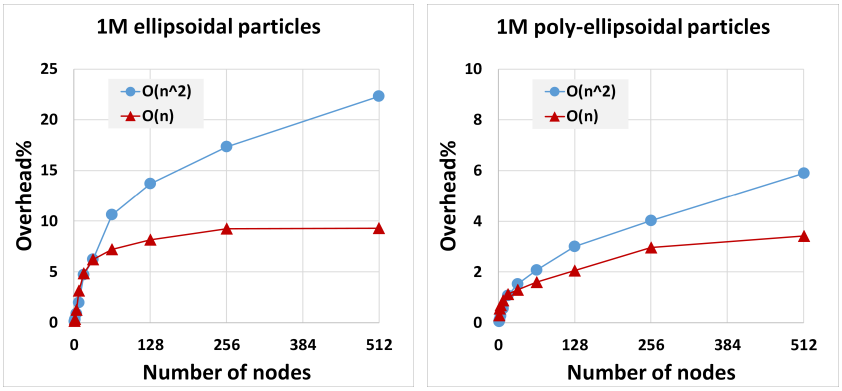
Figure 13: Parallel overhead percentage for ellipsoid and poly-ellipsoid.

On the scales of 150k and 1M particles plotted in Figure 12(c, d), $O(n^2)$ algorithm is slower than $O(n)$ with a few compute nodes, however it becomes faster than $O(n)$ with appropriate number of compute nodes used in practice. For example, $O(n^2)$ algorithm is slower than $O(n)$ using 1 node (9,375 PPC) to 2 nodes (4,688 PPC) but faster using 4 nodes (2,344 PPC) to 128 nodes (73 PPC) for 150k particles, and $O(n^2)$ algorithm is slower than $O(n)$ using 1 node (62,500 PPC) to 16 nodes (3,906 PPC) but faster using 32 nodes (1,953 PPC) to 512 nodes (122 PPC) for 1M particles. These are clearly plotted with log-log graphs in Figure 12(c', d').

The following conclusion is drawn based on the statistics of complex-shaped particles such as ellipsoid and poly-ellipsoid: in parallel computing, $O(n^2)$ algorithm is inefficient at coarse CG, however it executes faster than $O(n)$ algorithm at fine CGs that are mostly employed in practice.

As a result, the parallel overhead percentage, defined as the ratio of communication overhead to total time (communication overhead + computation time), is affected. First, the influence of particle shape is plotted in Figure 13 for 1M particles, where poly-ellipsoid gives a lower parallel overhead percentage due to its more CPU consumption on particle computation, meanwhile it has nearly the same parallel communication overhead as ellipsoid.

Second, Figure 14 compares the influence of $O(n^2)$ vs $O(n)$ algorithm for 1M particles. At the commonly used CGs, i.e., where more than 32 compute nodes are used, the $O(n^2)$ algorithm gives a higher parallel overhead



(a) Ellipsoid. (b) Poly-ellipsoid.

Figure 14: Parallel overhead percentage by $O(n^2)$ and $O(n)$ algorithms.

percentage than the $O(n)$ algorithm. This appears to be counterintuitive since $O(n^2)$ algorithm consumes more CPU on computation and thus should have lower parallel overhead percentage. However it is correct: referring to Figure 11(c', d') and Figure 12(c', d'), the $O(n^2)$ algorithm actually executes faster and consumes less total time at such CGs, namely, its wall clock time decreases faster than communication overhead with respect to the $O(n)$ algorithm, which results in a higher parallel overhead percentage. This is actually another proof that the $O(n^2)$ algorithm performs better than the $O(n)$ algorithm at fine CGs that are mostly employed in practice.

6. $O(n^2)$ VS $O(n)$ INFLUENCE ON SUPERLINEAR SPEEDUP

In parallel computing, *speedup* is defined as the ratio between sequential execution time and parallel execution time, as shown in Eq. (2), and *efficiency*, a measure of processor utilization, is defined as speedup divided by the number of processors used, according to Eq. (3).

$$\text{speedup } \psi(n,p) \equiv \frac{\text{sequential execution time}}{\text{parallel execution time}} \leq \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\varphi(n)/p+\kappa(n,p)} \quad (2)$$

$$\text{efficiency } \varepsilon(n,p) \equiv \frac{\text{sequential execution time}}{p\times\text{parallel execution time}} = \frac{\psi(n,p)}{p} \quad (3)$$

where n is the problem size (number of particles), p is the number of processors, $\sigma(n)$ is the inherently serial portion of computation, $\varphi(n)$ is the

parallelizable portion of computation, and $\kappa(n,p)$ is the overhead of parallelization (communication operations and redundant computation).

Yan and Regueiro [2018c] have discovered strong superlinear speedup in large scale simulations of parallel 3D DEM for complex-shaped particles, and the larger the scale is, the stronger is the superlinear speedup. They reproduced the phenomena on multiple DoD supercomputers in strong scaling and weak scaling measurements, and pointed out that cache miss rate is sensitive to the memory consumption shrinkage per processor, and the last level cache (LLC) contributes most significantly to the strong superlinear speedup among all of the three cache levels of modern microprocessors.

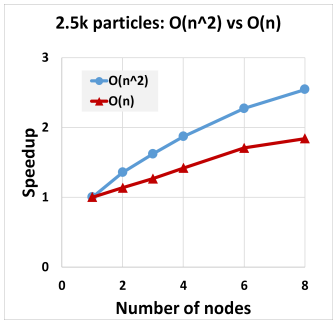
Their investigation was based on $O(n^2)$ neighbor search algorithm. On account of aforementioned statistics, it is imagined that $O(n^2)$ neighbor search algorithm could overestimate the speedup at coarse CGs although it does not at fine CGs. In this section, a comparison is made between $O(n^2)$ and $O(n)$ algorithms, to study if superlinear speedup still exists with $O(n)$ algorithm; and if it exists, to what degree it is reduced.

Figure 15 plots the speedup by $O(n^2)$ and $O(n)$ algorithms across 5 orders of magnitude of simulation scale, 2.5k, 12k, 150k, 1M and 10M ellipsoidal particles, each of which tests various number of compute nodes, i.e., various CGs. It is interesting to see that the $O(n)$ algorithm always exhibits a lower speedup than $O(n^2)$ algorithm across all CGs, although it executes faster with coarse CG and slower with fine CG than $O(n^2)$ for larger scale computation, as discussed in Section 5.

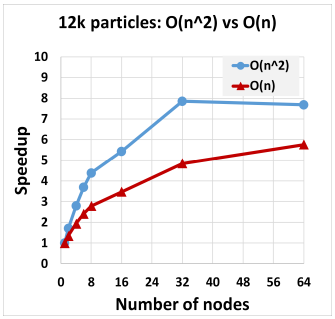
The speedup gap is due to the difference of base point measurement using 1 compute node: $O(n)$ algorithm runs much faster at such a coarse CG than $O(n^2)$ algorithm. As a result, its speedup cannot go as high as that of $O(n^2)$ algorithm. The explanation is even clearer on the grounds of data listed in Table 6: the wall clock time ratio between $O(n^2)$ and $O(n)$ algorithms is 1.49 with 1 node, and it decreases to 0.87 with increasing number of nodes.

Figure 16 plots the efficiency by $O(n^2)$ and $O(n)$ algorithms across 4 orders of magnitude of simulation scale. It is expected that the $O(n)$ algorithm always exhibits a lower efficiency than $O(n^2)$ algorithm across all CGs. It is shown that neither $O(n^2)$ nor $O(n)$ reveals superlinear speedup (greater-than-1 efficiency) on the scales of 2.5k and 12k particles. At 150k particles, $O(n^2)$ exhibits superlinear speedup whereas $O(n)$ does not. On the scales of 1M and 10M particles, both $O(n^2)$ and $O(n)$ present strong superlinear speedup.

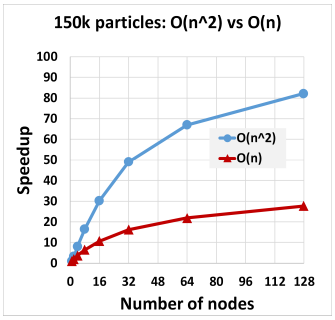
The average speedup or efficiency ratio between $O(n^2)$ and $O(n)$ algo-



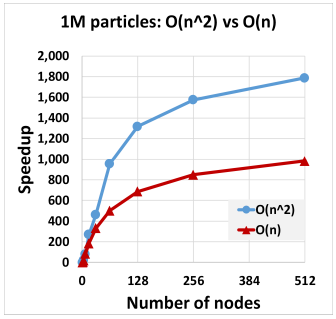
(a) 2.5k particles.



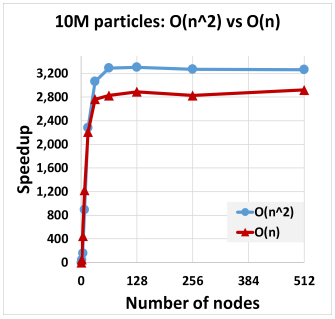
(b) 12k particles.



(c) 150k particles.



(d) 1M particles.



(e) 10M particles.

Figure 15: Speedup in computing ellipsoids across 5 orders of magnitude of simulation scale.

Table 6: $O(n^2)$ vs $O(n)$ at 1M ellipsoidal particles

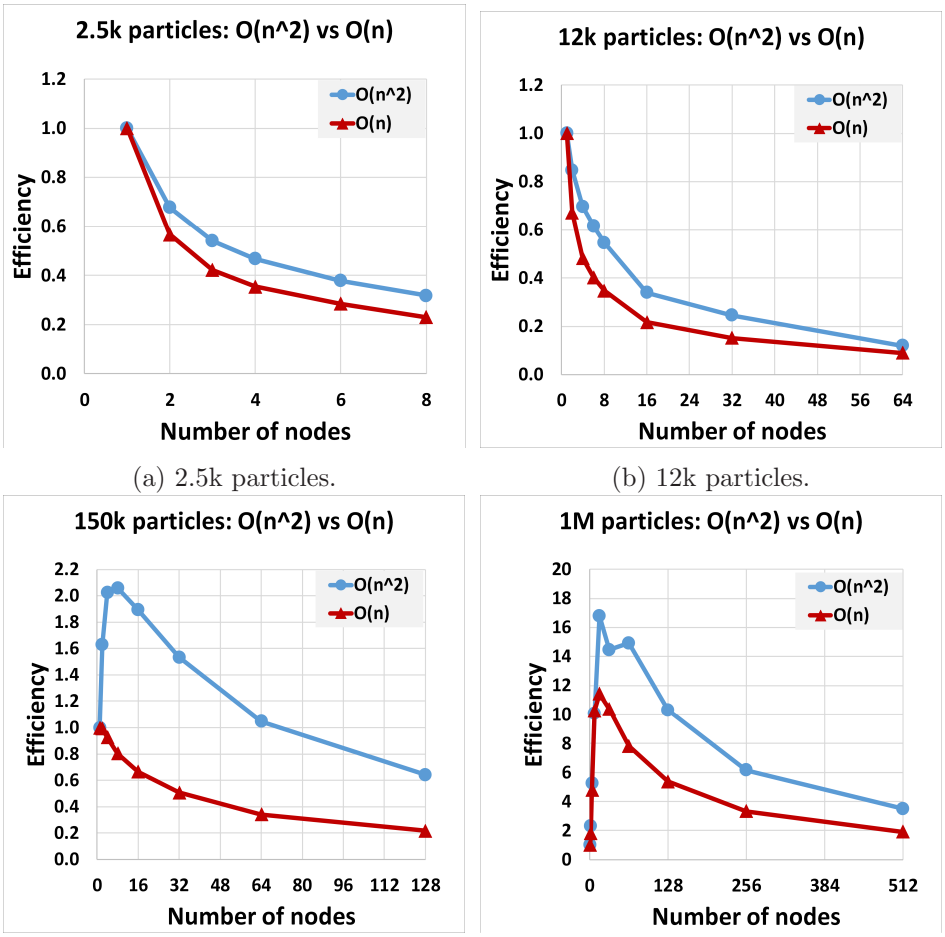
	wall clock time (s)			speedup			efficiency		
nodes	$O(n^2)$	$O(n)$	ratio	$O(n^2)$	$O(n)$	ratio	$O(n^2)$	$O(n)$	ratio
1	128284.2	85997.2	1.49	1.0	1.0	1.00	1.0	1.0	1.00
2	27783.8	23770.4	1.17	4.6	3.6	1.28	2.3	1.8	1.28
4	6135.4	4502.4	1.36	20.9	19.1	1.09	5.2	4.8	1.09
8	1597.1	1048.0	1.52	80.3	82.1	0.98	10.0	10.3	0.98
16	477.2	470.1	1.02	268.8	182.9	1.47	16.8	11.4	1.47
32	277.3	259.3	1.07	462.6	331.7	1.39	14.5	10.4	1.39
64	134.3	171.9	0.78	955.3	500.3	1.91	14.9	7.8	1.91
128	97.3	125.4	0.78	1318.3	686.0	1.92	10.3	5.4	1.92
256	81.4	101.3	0.80	1575.2	848.6	1.86	6.2	3.3	1.86
512	71.7	87.6	0.82	1788.2	982.2	1.82	3.5	1.9	1.82
768	83.1	95.3	0.87	1544.5	902.0	1.71	2.0	1.2	1.71
1024	71.6	82.5	0.87	1792.9	1042.5	1.72	1.8	1.0	1.72
average			1.0			1.5			1.5

rithms is 1.5 for 1M particles according to Table 6, in other words, the speedup in $O(n)$ algorithm is reduced by approximately 1/3 relative to $O(n^2)$ algorithm, even though both algorithms reveal suplinear speedup.

7. WEAK SCALING MEASUREMENT

The Performance Application Programming Interface (PAPI) is coded to measure the FLOPS performance based on weak scaling tests, where the problem size (workload) assigned to each processor stays constant; namely, a constant CG is employed across different orders of magnitude of simulation scale. In these tests, 2,500 complex-shaped particles are assigned to each compute node (approximately 150 particles per core); the number of compute nodes ranges from 1 to 2,048, and the number of particles ranges from 2,500 to 5,120,000 accordingly; the $O(n^2)$ algorithm is used since it performs better than the $O(n)$ algorithm at fine CGs that are mostly employed in practice. Hereby it is emphasized again that the CG of 150 particles per core is optimized for complex-shaped particles, not for spheres.

Figure 17(a) and (b) plots the total and per core FLOPS performance with regard to problem size (in direct proportion to number of compute nodes), respectively. It can be seen clearly that the total FLOPS exhibits



(a) 2.5k particles.

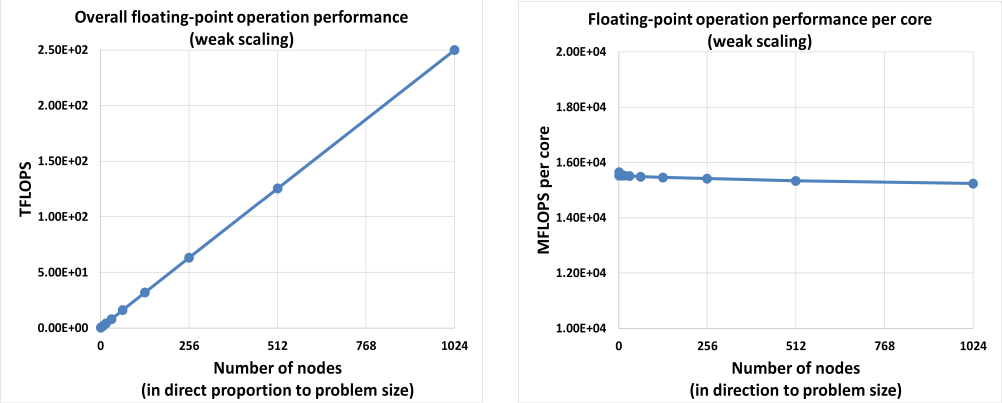
(b) 12k particles.

(c) 150k particles.

(d) 1M particles.

Figure 16: Efficiency in computing ellipsoids across 4 orders of magnitude of simulation scale.

a linear relationship with regard to the problem size, and the uniprocessor FLOPS performance nearly stays constant in spite of the problem size change.



(a) Overall FLOPS. (b) Per core FLOPS.

Figure 17: Floating-point operation performance on Spirit (weak scaling).

Figure 18 plots the module execution times. The communication time stays constant until the number of nodes reaches 512 or 1,024, whereby the increase is most likely attributed to the lack of hypercube interconnect [Rudi et al., 2015].

The computation time increases slightly with respect to the increase of number of compute nodes, yet overall it exhibits linear scaling with close to constant computation time. The slight increase of computation time is attributed to the characteristics of memory consumption and cache hit/miss rate on the system [Yan and Regueiro, 2018a].

8. SUMMARY

This paper presents implementation comparison between $O(n^2)$ and $O(n)$ neighbor search algorithms, studies their effects for different particle shape complexity and computational granularity (CG) in serial computing, analyzes the performance difference in serial and parallel computing, and investigates the influence on superlinear speedup of 3D DEM for complex-shaped particles.

In complex-shaped DEM, the neighbor search algorithm or time complexity only affects the performance of neighbor estimate, and they have no

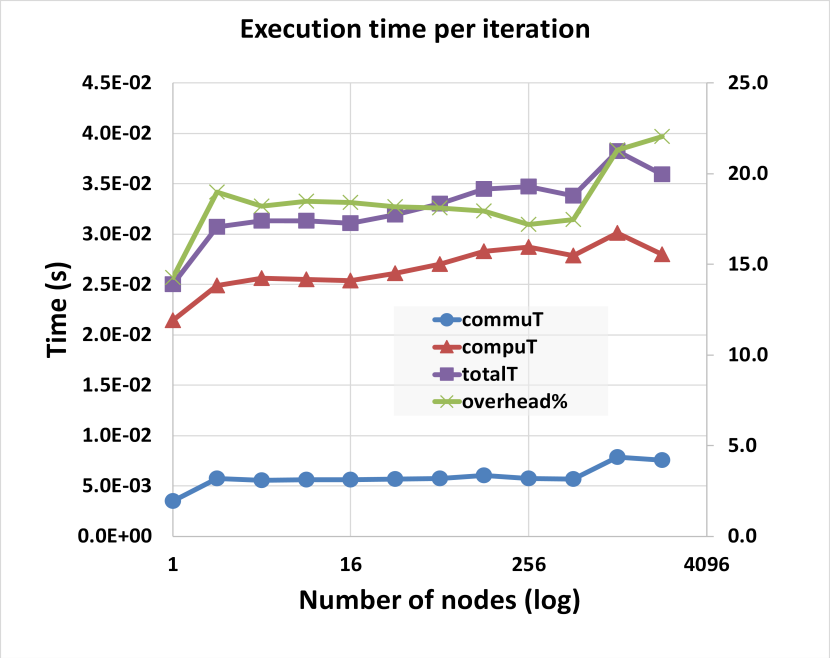


Figure 18: Weak scaling measurement on Spirit.

bearing on contact resolution, which usually takes a significant majority of overall floating point operations.

Serial computing reveals that the more complex the particle shapes (from sphere to ellipsoid to poly-ellipsoid), the smaller the neighbor search fraction (NSF); and the lower the computational granularity (CG), the smaller the NSF.

In serial computing, the $O(n)$ search algorithm works more efficiently than $O(n^2)$ on the whole, especially for situations with coarser CG; however the $O(n)$ algorithm is inferior to $O(n^2)$ for CG finer than the parallel optimum, owing to its particle assignment overhead.

In parallel computing of complex-shaped 3D DEM, the $O(n^2)$ algorithm is inefficient at coarse CG, however it executes faster than the $O(n)$ algorithm at fine CGs that are mostly employed in computational practice, which means that $O(n^2)$ algorithm outperforms $O(n)$ algorithm in large-scale parallel 3D DEM simulations generally.

Both $O(n^2)$ and $O(n)$ algorithms exhibit a strong superlinear speedup on large scale simulations of complex-shaped 3D DEM. The $O(n)$ algorithm

always exhibits a lower speedup than the $O(n^2)$ algorithm across all computational scales and granularities, mostly due to the base point measurement at 1 compute node, whereby the $O(n)$ algorithm executes much faster than the $O(n^2)$ algorithm. On average, the speedup in $O(n)$ algorithm is reduced by approximately 1/3 relative to $O(n^2)$ algorithm on the simulation scale of 1 million ellipsoidal particles.

The weak scaling measurements of complex-shaped particles show a linear relationship between FLOPS and the number of compute nodes, and close-to-linear scalability in terms of nearly constant computation time.

Acknowledgments

We would like to acknowledge the support provided by ONR MURI grant N00014-11-1-0691, and the DoD High Performance Computing Modernization Program (HPCMP) for granting us the computing resources required to conduct this work. We declare that there is no conflict of interest.

References

Gary W Delaney, Paul W Cleary, Matt D Sinnott, and Rob D Morrison. Novel application of dem to modelling comminution processes. In *IOP Conference Series: Materials Science and Engineering*, volume 10, page 012099. IOP Publishing, 2010.

Gary S Grest, Burkhard Dünweg, and Kurt Kremer. Vectorized link cell fortran code for molecular dynamics simulations for a large number of particles. *Computer Physics Communications*, 55(3):269–285, 1989.

Heinrich Hertz. Ueber die Berührung fester elastischer Körper. [On the fixed elastic body contact]. *Journal für die reine und angewandte Mathematik (Crelle)*, 92:156–171, 1882.

Hosagrahar V Jagadish, Beng Chin Ooi, Kian-Lee Tan, Cui Yu, and Rui Zhang. idistance: An adaptive b+-tree based indexing method for nearest neighbor search. *ACM Transactions on Database Systems (TODS)*, 30(2): 364–397, 2005.

Keng-Wit Lim and José E Andrade. Granular element method for three-dimensional discrete element calculations. *International Journal for Numerical and Analytical Methods in Geomechanics*, 38(2):167–188, 2014.

R.D. Mindlin. Compliance of elastic bodies in contact. *Trans. ASME, J. App. Mech.*, 16(3):259–268, 1949.

R.D. Mindlin and H. Deresiewicz. Elastic spheres in contact under varying oblique forces. *Trans. ASME, J. App. Mech.*, 20(3):327–344, 1953.

Marius Muja and David G Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 2:331–340, 2009.

A. Munjiza and K.R.F. Andrews. Nbs contact detection algorithm for bodies of similar size. *International Journal for Numerical Methods in Engineering*, 43(1):131 – 149, 1998.

John F. Peters, Mark A. Hopkins, Raju Kala, and Ronald E. Wahl. A polyellipsoid particle for nonspherical discrete element method. *Engineering Computations*, 26(6):645–657, 2009.

Johann Rudi, A Cristiano I Malossi, Tobin Isaac, Georg Stadler, Michael Gurnis, Peter WJ Staar, Yves Ineichen, Costas Bekas, Alessandro Curioni, and Omar Ghattas. An extreme-scale implicit solver for complex pdes: highly heterogeneous flow in earth's mantle. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 5. ACM, 2015.

Christian Wellmann, Claudia Lillie, and Peter Wriggers. A contact detection algorithm for superellipsoids based on the common-normal concept. *Engineering Computations*, 25(5):432–442, 2008.

John R Williams and Alex P Pentland. Superquadrics and modal dynamics for discrete elements in interactive design. *Engineering Computations*, 9 (2):115–127, 1992.

John R Williams, Eric Perkins, and Ben Cook. A contact algorithm for partitioning n arbitrary sized objects. *Engineering Computations*, 21(2/3/4): 235–248, 2004.

Beichuan Yan and Richard Regueiro. Large-scale dynamic and static simulations of complex-shaped granular materials using parallel three-dimensional discrete element method (dem) on dod supercomputers. *Engineering Computations*, (just-accepted):00–00, 2018a.

Beichuan Yan and Richard A Regueiro. A comprehensive study of mpi parallelism in three-dimensional discrete element method (dem) simulation of complex-shaped granular particles. *Computational Particle Mechanics*, pages 1–25, 2018b.

Beichuan Yan and Richard A Regueiro. Superlinear speedup phenomenon in parallel 3d discrete element method (dem) simulations of complex-shaped particles. *Parallel Computing*, 2018c.

Beichuan Yan, Richard A Regueiro, and Stein Sture. Three-dimensional ellipsoidal discrete element modeling of granular materials and its coupling with finite element facets. *Engineering Computations*, 27(4):519–550, 2010.