Superlinear Speedup Phenomenon in Parallel 3D Discrete Element Method (DEM) Simulations of Complex-shaped Particles

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

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

Abstract

Strong superlinear speedup has been discovered in large scale simulations of parallel 3D DEM for complex-shaped particles, which is based on an algorithm of spatial domain decomposition, and exhibits the "high-CPU-low-memory" characteristics. The interpretation of this phenomenon requires a careful examination of the speedup theory and practice in the field of parallel computing. The superlinear speedup is investigated from three perspectives: (i) memory footprint per process, (ii) cache miss rates of L1, L2 and L3 level caches, and (iii) uniprocessor performance, using a wide range of problem size (across five orders of magnitude of simulation scale regarding number of particles) and number of compute nodes (1 to 2,048 nodes) on DoD supercomputers. The Performance-API (PAPI) is employed in the source code to measure cache miss rate and FLOPS. The strong scaling measurements show 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, and this is also revealed in the weak scaling measurements. The findings are associated with the inherently perfect scalability of 3D DEM: its memory scalability function is a nonlinearly decreasing function of the number of processors. In addition, a constant (non-increasing) uniprocessor FLOPS performance w.r.t problem size can also contribute to the superlinear speedup.

The superlinear speedup is a common phenomenon for large scale 3D

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

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

DEM simulations of complex-shaped particles, and the larger the scale, the stronger is the superlinear speedup. DEM researchers should take advantage of this effect to speedup their parallel simulations.

Keywords: superlinear speedup, parallel discrete element method, granular materials, complex-shaped, cache miss rate, FLOPS

---

1. INTRODUCTION TO SUPERLINEAR SPEEDUP

1.1. What is superlinear speedup?

In parallel computing, *speedup* is defined as the ratio between sequential execution time and parallel execution time, as shown in Eq. (1), and *efficiency*, a measure of processor utilization, is defined as speedup divided by the number of processors used, according to Eq. (2).

$$\text{speedup } \psi(n,p) \equiv \frac{\text{sequential execution time}}{\text{parallel execution time}} \leq \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\varphi(n)/p+\kappa(n,p)} \quad (1)$$

$$\text{efficiency } \varepsilon(n,p) \equiv \frac{\text{sequential execution time}}{p\times\text{parallel execution time}} = \frac{\psi(n,p)}{p} \quad (2)$$

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).

The term "superlinear speedup" has been used to describe a computation using p processors which is more than p times faster than the same computation performed on a uniprocessor (Wilkinson and Allen, 1999). It is equivalent to the notion of "greater than 100% or 1 efficiency".

It should be noted that the term p in Eqs. (1) and (2) could have different meaning although it was originally referred to the number of processors. The microprocessor architecture has entered the multicore era, and a processor/CPU (a socket on a computer motherboard) contains multiple cores, so p can refer to the number of cores; contemporary supercomputers may have thousands or ten thousands of compute nodes, each of which integrates one or more multicore microprocessors, then p can refer to the number of compute nodes for those large scale computational tasks. We distinguish the node speedup/efficiency from the core speedup/efficiency in this paper.

1.2. History of parallel speedup research

To better understand the superlinear speedup phenomenon, it is necessary to review the history of how researchers have recognized the parallel speedup, which unavoidably involves the superlinear speedup.

In 1967 the Amdahl's law was put forward based on a general observation about the performance improvement limitation (Amdahl, 1967). The law states that the maximum speedup ψ achievable by a parallel computer with p processors is

$$\psi \leq \frac{1}{f + (1-f)/p}, \tag{3}$$

where $f = \sigma(n)/(\sigma(n) + \varphi(n))$ denotes the fraction of operations in a computation that must be performed sequentially. Amdahl's law indicates that the serial portion, which cannot be improved by parallelization, will quickly dominate the performance, and further improvement of the improvable or parallelizable portion will have little effect. It was used as an argument against massively parallel processing.

In 1988, Gustafson-Barsis's Law was used to justify massively parallel processing (MPP), which states that the maximum speedup (also called scaled speedup) ψ achievable by a parallel program is

$$\psi \leq p + (1-p)s, \tag{4}$$

where $s = \sigma(n)/(\sigma(n) + \varphi(n)/p)$ denotes the fraction of time spent in the parallel computation performing inherently sequential operations. Gustafson (1988) pointed out that: (1) when serial fraction is very low, speedup could be very high and forms an unforgivingly steep function of s near $s = 0$. He achieved a speedup of 1021, 1020 and 1016 for different applications, respectively, on a 1024-processor hypercube (Gustafson et al., 1988); (2) as an approximation, only the parallel part of a program scales with the problem size; (3) it is important for the research community to overcome "mental block" against massive parallelism.

Sun and Ni (1990, 1993) studied the fixed-size speedup, fixed-time speedup, and memory-bounded speedup, and derived corresponding speedup formulations. The memory-bounded speedup is also known as Sun and Ni's Law. They pointed out that the simplified fixed-size speedup is Amdahl's law, and the simplified fixed-time speedup is Gustafson's scaled speedup.

Sun and Gustafson (1991) proposed two new performance metrics: (1) sizeup, which provides a "fair" performance measurements; and (2) the generalized speedup, which emphasizes that speedup is the ratio of speeds, not

times.

$$\text{sizeup} = \frac{\text{parallel work}}{\text{sequential work}}, \tag{5}$$

$$\text{generalized speedup} = \frac{\text{parallel execution speed}}{\text{sequential execution speed}}. \tag{6}$$

The authors also studied their relations and pointed out that: speedup is the restriction of generalized speedup to fixed work, and sizeup is the restriction of generalized speedup to fixed time. It is worth noting that performance metric by generalized speedup avoids running large problem size on a single node, i.e., the sequential speed is measured with a small problem size (appropriate for one node processing), and parallel speedups are measured with large problem sizes (appropriate for large number of nodes).

Shi (1996) revealed that Amdahl's Law and Gustafson's Law are in fact identical by establishing the mathematical equivalence between them. The author rigorously distinguished scaled percentage s and non-scaled percentage f of the serial fraction of a program, and derived that

$$f = \frac{1}{1 + \frac{(1-s)p}{s}}. \tag{7}$$

He gave an impressive example of translating a scaled serial fraction of 0.4 to 0.8 percent in Gustafson (1988) to a non-scaled serial fraction of 0.0004 to 0.0008 percent, respectively. The author also pointed out that using Amdahl's Law as an argument against massively parallel processing is not valid, and it is because f can be very close to zero for many practical applications, thus very high speedups are possible using massively many processors.

In 2008, when the microprocessor architecture entered multicore era, Hill and Marty (2008) presented a pessimistic view of multicore scalability based on their analysis using Amdahl's law and challenged readers to develop better models.

Sun and Chen (2010) studied multicore scalability under fixed-time and memory-bound conditions from the data access perspective, and concluded that multicore architectures are fundamentally scalable and not limited by Amdahl's law, and that multicore architectures are capable of extensive scalability. The authors also pointed out that the need for technical improvements is primarily in memory performance.

1.3. Lack of superlinear speedup study

The study of superlinear speedup in parallel computing is limited according to the literature. Faber et al. (1986) stated that superlinear speedup of an efficient sequential algorithm is not possible, based on an assumption of hardware limitation and context switching time, etc.

Karp and Flatt (1990) found a decreasing experimentally determined serial fraction (EDSF) on a 4-processor Convex C-240 and associated it with a possible "superlinear" speedup. When he studied the winners of the Gorden Bell Awards in 1987 (`http://www.sc2000.org/bell/pastawrd.htm`), he found that all of the three problems (Beam Stress Analysis, Surface Wave Simulation, and Unstable Fluid Flow Model) reveal a significant reduction of the EDSF using 4 to 1,024 processors, and he attributed the significantly decreasing EDSF to "superlinear" speedup.

Helmbold and McDowell (1990) mentioned several apparent sources of superlinear speedup: hidden memory latency, subdivision of system overhead, and randomized algorithms.

Gustafson (1990) pointed out: *" Superlinear speedup can result whenever problem size per processor is reduced, whether from fixed sized or fixed time performance evaluation, such that $U(n)$ crosses regimes and appears decreasing instead of increasing. The decrease must be enough that the usual sources of parallel inefficiency (load imbalance, serial algorithm steps, interprocessor communication) are compensated."*, whereby $U(n)$ denotes the uniprocessor performance (MFLOPS) as a function of problem size n.

The author also pointed out two new sources of superlinear speedup: the different speeds of memory inherent in distributed memory ensembles, and the shift in time fraction spent on different-speed tasks.

The author stated with great foresight that *" tiered memory can make performance increase instead of decrease as problem size per processor shrinks, and workload can shift to routines with higher speed as the problem is scaled. Superlinear speedup results in such cases. Superlinear speedup, far from being an anomaly, becomes commonplace when the performance model makes realistic assumptions about memory speed and problem scaling."*

Sienicki et al. (1994) divided superlinear speedup problems into two types: (1) search problems, whereby some paths leading to wrong results in parallel processing may be eliminated earlier than that in sequential processing (Moldovan, 2014); and (2) computing problems. The authors thought that *"the superlinear speedup is somewhat limited to small number of processors and may not be scalable"*.

Shi (1996) stated that the theoretically linear and superlinear speedups are not possible because the serial percentage in Amdahl's Law is never zero in practice. Nevertheless, there are two factors that can be used to produce linear or superlinear speedups in practice,: (1) use of a resource-constrained serial execution as the base for speedup calculation; (2) and use of a parallel implementation that can bypass large number of calculation steps while yielding the same output of the corresponding serial algorithm.

The author defined two different types of sequential algorithms: structure persistent (SP) and non-structure persistent (NSP), and stated that superlinear speedup is only possible for NSP algorithms.

Nagashima et al. (1995) observed superlinear speedup when the number of processors is appropriate to the size of problem on a distributed memory parallel processing system with cache architecture, during the calculation of density of states (DOS) of cyclic polyacenes using a proper numerical method and message passing library. The authors considered the reason for the superlinear speedup to be avoiding the neck of memory architecture such as cache miss rate.

Wilkinson and Allen (1999) thought the superlinear speedup is usually due to using a suboptimal sequential algorithm, a unique feature of the system architecture that favors the parallel formation, or an indeterminate nature of the algorithm. However, they admitted that a common reason for superlinear speedup is that a multiprocessor system has extra memory, which can hold more of the problem data at any instant and lead to less disk memory traffic.

Ristov and Gusev (2012) obtained a superlinear speedup with efficient cache exploitation on a shared memory multiprocessor for matrix multiplication algorithm, which is SP according to Shi (1996). The experiments were carried out on a workstation of one AMD 4-core Phenom processor and a workstation of four 4-core AMD Opteron processors. Both processors have three levels of cache. The author found that there is a *superlinear region* (defined as when memory requirement fits in a cache) of the problem size where the normalized performance per core for parallel execution is better than in sequential execution, and he obtained a superlinear speedup beyond the limits specified in Gustafson's law.

On the SCinet Wiki page (`https://wiki.scinet.utoronto.ca/wiki/index.php/Introduction_To_Performance`), it states that "*It isn't uncommon to achieve greater than 100% parallel efficiencies for small numbers of processors for some types of problems; as you go to more processors, you also*

have more processor cache, and thus more of the problems data can fit into fast cache. This is called super-linear speedup and sadly seldom extends out to very many processors."

1.4. Outline of this paper

This paper presents performance measurement and analysis, particularly the superlinear speedup phenomenon, in simulating granular/geotechnical materials using complex-shaped 3D DEM across five orders of magnitude of simulation scale (number of particles). The paper contains eleven sections. Section 1 has reviewed the research history of parallel computing speedup and the study in superlinear speedup phenomenon; section 2 is an introduction to the DEM method, its computational features and recent efforts in its parallelism; section 3 covers several important concepts and techniques in the design of parallelism of 3D DEM; section 4 derives the iso-efficiency relation for 3D DEM computation and serves as an theoretical tool to interpret the speedup phenomenon; section 5 introduces three targeted supercomputers used in this work and particularly the microprocessor architecture on these systems; section 6 populates the performance data measured from the three supercomputers and describes the difference between node and core speedup/efficiency; section 7 analyzes the potential causes of the superlinear speedup of 3D DEM; section 8 implements and performs cache miss (per second) measurements using Performance-API (PAPI), and tries to explain the relationship between cache miss (per second) and superlinear speedup; section 9 evaluates the weak scaling measurements of 3D DEM; section 10 cites the conclusion (Yan and Regueiro, 2018a) on the influence of different neighbor search algorithms on the computational performance; and the last section gives conclusion and outlook.

2. THE DEM AND ITS PARALLELISM

The Discrete Element Method (DEM) has been applied to study the mechanical and micro-mechanical behavior of particle assemblages for more than 40 years since its introduction in the late 1970s by Cundall and Strack (1979). However, application of 3D DEM to simulating practical problems involving granular and geomechanical materials is still limited in terms of problem size (namely, number of particles). For example, most applications involving complex-shaped particles such as axi-symmetric/revolution ellipsoids (Ng, 1994, 2004) or true ellipsoids (Yan et al., 2010) constrain their

number of particles to a few thousand. This is mainly due to the fact that the DEM poses high computational demands characterized by CPU-intensive interparticle contact detection and explicit time integration schemes.

Under many situations, the length scale and problem size of practical engineering problems cannot be circumvented even if a multi-scale model is employed. For instance, to study the impact of blast waves on gravitationally deposited coarse-grained soils in which an explosive charge is ignited, a 40 cm x 40 cm x 40 cm specimen composed of ellipsoidal sand particles contains approximately 500 million 0.1$\sim$1 mm diameter particles depending on the particle shapes and size distribution. The limitation of problem size poses a barrier to simulating many science phenomena, engineering problems and laboratory tests involving complex-shaped particles that are the same size as physical particles, such as sand dune movement or sediment transport in geomorphology, quick sand phenomenon and soil liquefaction in earthquake engineering, soil-tire interaction in Mars Exploration Rover Mission, landslide and its expansion in geotechnical engineering, precast pile installation/penetration mechanisms and static/dynamic testing in civil engineering, internal shear band evolution in soil mechanics (Fu and Dafalias, 2011a,b), etc.

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. A typical procedure of DEM analysis (Yan et al., 2010) consists of three major computational steps in sequence, which are integrated in time using central difference method until a simulation is completed:

- contact detection between particles, including two phases:
    1. *neighbor search (neighbor estimate)*
    2. *geometric contact resolution*
- contact force computation for each pair of particles in contact.
- 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: 1. the neighbor search (or spatial reasoning) phase, and 2. the geometric 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. 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.

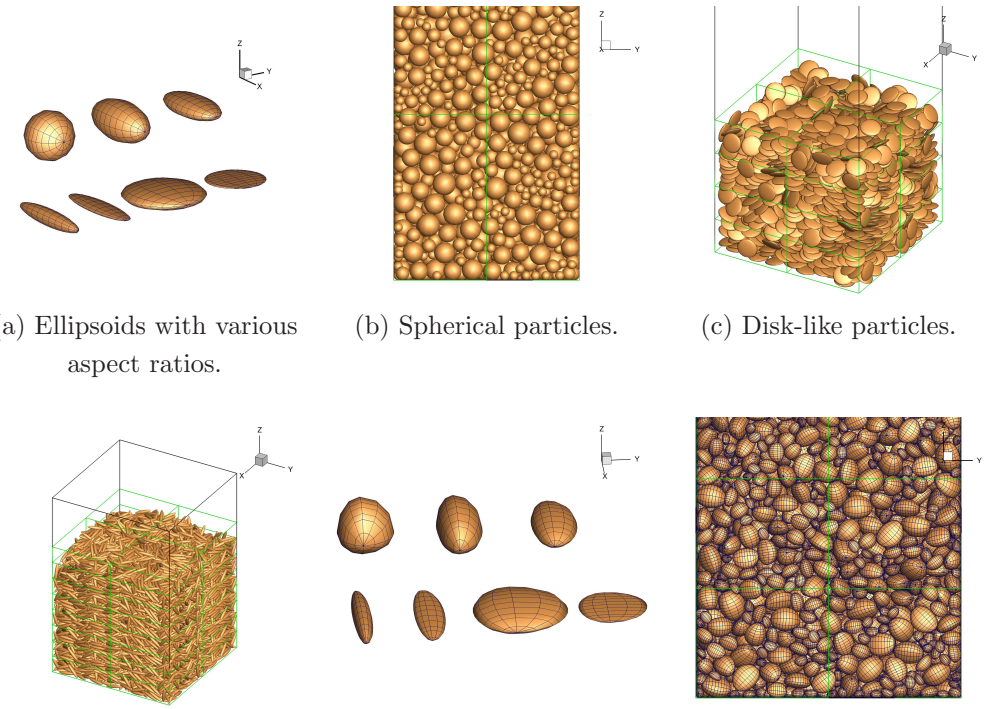
2.2. Neighbor search

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 method* (Munjiza and Andrews, 1998; Williams et al., 2004) or link-cell (LC) method (Grest et al., 1989), where n denotes the number of particles.

The binning method and the LC method are essentially the same, and they are just different names used in geomechanics and molecular dynamics (MD), respectively. 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 *no binary search (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.

2.3. Geometric 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 three-axis 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 force, and a

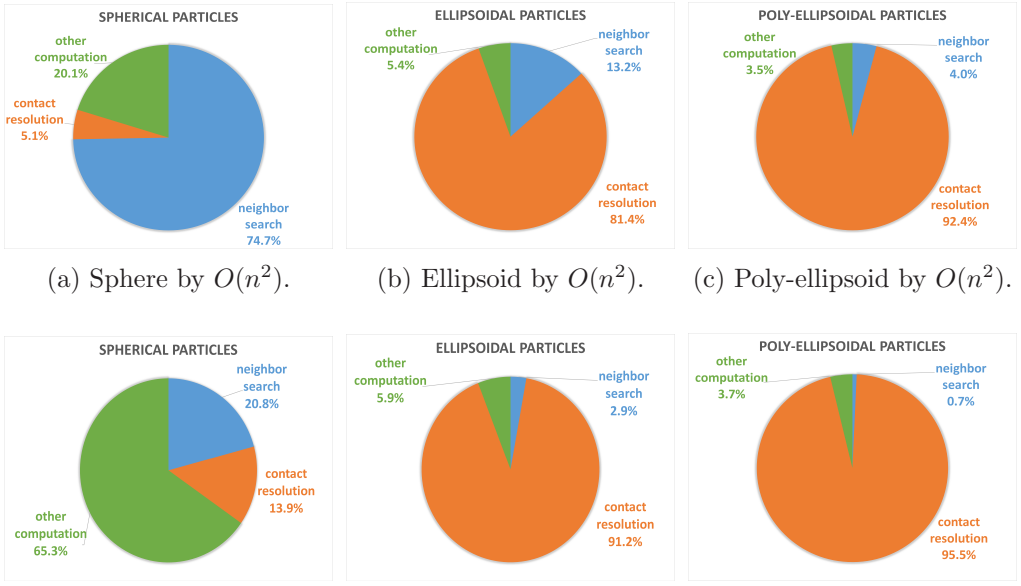
low-precision solver can lead to numerical instability or spurious explosion of particles. Therefore, an iterative eigenvalue method, which performs QR-decomposition of real Hessenberg matrices, is selected to find roots of the polynomial and determine the contact geometry. The algorithm and its implementation 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, as shown in Figure 1(e∼f). It is worth emphasizing that the simulations of 3D DEM discussed in the paper are mainly focused on complex-shaped particles such as true 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 the simplistic spheres. 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.

2.4. Weight of neighbor search

It is worth noting that the three neighbor search algorithms, $O(n^2)$, $O(n\log n)$ and $O(n)$, only affect the performance of neighbor search. They have no bearing on contact resolution. It can be imagined that the overall performance improvement resulting from these algorithms might be highly limited for complex-shaped particles, because neighbor search only takes up a small fraction of floating point operations in the whole computation. For instance, contact resolution between a pair of three-axis ellipsoids is approximately 35 times as expensive as that of a pair of spheres, and contact resolution between a pair of poly-ellipsoids is nearly 260 times as expensive as that of a pair of spheres (Yan and Regueiro, 2018a).

Both $O(n^2)$ and $O(n)$ algorithms are implemented and tested in the paper. Figure 2 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 neighbor search fraction (NSF) is, whereby NSF is defined as the ratio of neighbor search time to the total



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

contact detection time. With poly-ellipsoid computing by $O(n)$ algorithm, the NSF is as low as 0.7% whereas contact resolution takes up to 95.5%. Furthermore, the lower the computational granularity (CG), i.e., number of particle per process, the smaller the NSF.

Yan and Regueiro (2018a) pointed out: in both serial and parallel computing of complex-shaped 3D DEM, the $O(n^2)$ neighbor search 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.

2.5. Efforts in developing parallel DEM

There has been considerable efforts in developing parallel DEM codes in recent years. Baugh Jr and Konduri (2001) presented a distributed computing system for DEM that is designed for loosely-coupled networks of workstations using low-level BSD (Berkeley Software Distribution) sockets. The implementation is used to simulate as many as 200,000 spherical particles using eight processors. As an example, the speedup is close to 6 using 8 processors for the computation of 120k spherical particles.

Washington and Meegoda (2003) simulated a triaxial test using an algorithm titled "TRUBAL for Parallel Machines (TPM)" and showed its benefits over the serial version DEM code, TRUBAL. Two simulation scales are tested and compared, one with 403 spheres and the other with 1,672 spheres, on Connection Machine (CM-5) with 512 nodes at the Pittsburgh Supercomputing Center. The speedup is as low as 7.9 using 512 nodes for 403 spheres.

Maknickas et al. (2006) described the DEMMAT_PAR code for simulation of visco-elastic frictional granular media, which has been created in the Parallel Computing Laboratory of Vilnius Gediminas Technical University. The parallel performance tests were carried out on their PC cluster VILKAS for systems consisting of 5k, 20k and 100k particles whose parameters are artificially assumed, for example, the Young's modulus is as low as 1 MPa. The speedup is approximately 11 on 16 processors for 100k spherical particles.

Henty (2000) chose to investigate the performance of a much smaller test code that implements precisely the same algorithm but has limited functionality rather than tackle a complete physics DEM application with all of its functionality and complexity. He implemented a hybrid parallelization of the message-passing and shared-memory models simultaneously.

Vedachalam and Virdee (2011) used LAMMPS (large-scale atomic and molecular massively parallel simulator) and LIGGGHTS (LAMMPS improved for general granular and granular heat transfer simulations) to study the motion of snow particles, wherein the snow grains are assumed to be spherical particles of 5 mm diameter. An empirical coefficient of restitution (ratio of rebound velocity to impact velocity) is adopted rather than the strict Hertz nonlinear contact model, while Mindlin's history-dependent shear model is not considered. As for the performance gain of parallelism, the authors wrote "on 480 processors for 75K particles, the speedup was 1.99, while on 960 processors for same number of particles speedup achieved was 2.52" with regard to 120 processors, which is a relatively low performance gain.

Some of these works have achieved fairly good speedup, nevertheless, none of them has been able to capture superlinear speedup phenomenon. This is probably owing to the following reasons:

- simulating at relatively small order of magnitude of simulation scale.
- focusing on simplistic spherical particles rather than complex-shaped particles which pose much higher CPU demand.
- lack of computational resources to measure performance of large scale

simulations.

3. PARALLELISM DESIGN OF 3D DEM

This section covers the most important concepts and techniques used in the design of 3D DEM parallelism.

3.1. Link-block and four-step MPI design

The concept of link-block (LB) is put forward for design of the parallel algorithm, illustrated in Figure 3. With introduction of LB, the Foster's four-step methodology (Foster, 1995) can be readily applied:

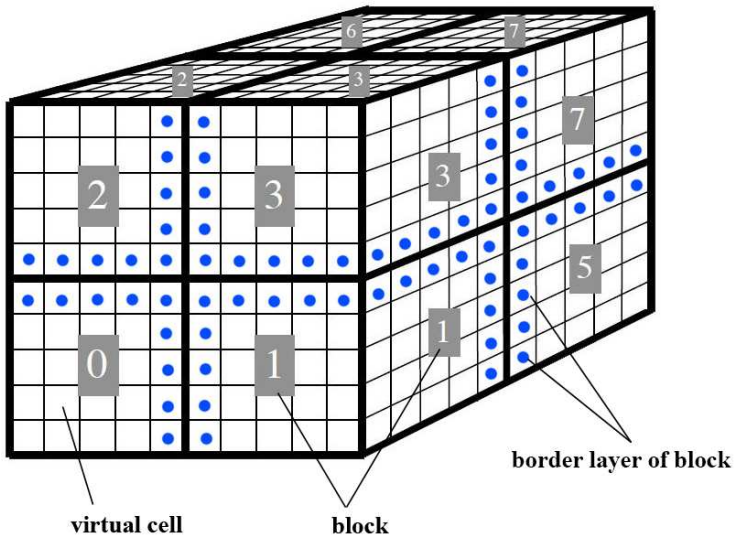


Figure 3: Schematic of link-blocks, virtual cells and border layers.

Partitioning: The computational domain is divided into blocks; each block may consist of many virtual cells. The size of the virtual cells may be 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 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 shown in Figure 3.

Mapping: Assignment of agglomerated tasks to processors, which could be to a core, multicores, a CPU, multiCPUs within a node, or even a whole node.

3.2. Flowchart of the parallel algorithm

The flowchart of the parallel code is designed as in Figure 4. It contains twelve flow processes or steps, among which one is sequential, two are partially parallel, and nine fully parallel. Ten of the twelve processes are repeated in the iteration loop until the computation is completed.

In comparison to the serial algorithm, the parallel one 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 part 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 it does not execute in each iteration loop, 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.

Among the twelve steps, most of them only involve local communication while three of them are associated with global communication.

3.3. Interblock communication

The interblock communication is illustrated by a particle assemblage composed of two layers of 81 randomly-sized particles, which are gravitationally deposited into a rigid container. The top view is shown in Figure 5, wherein the container is partitioned into four blocks separated by blue dash lines.

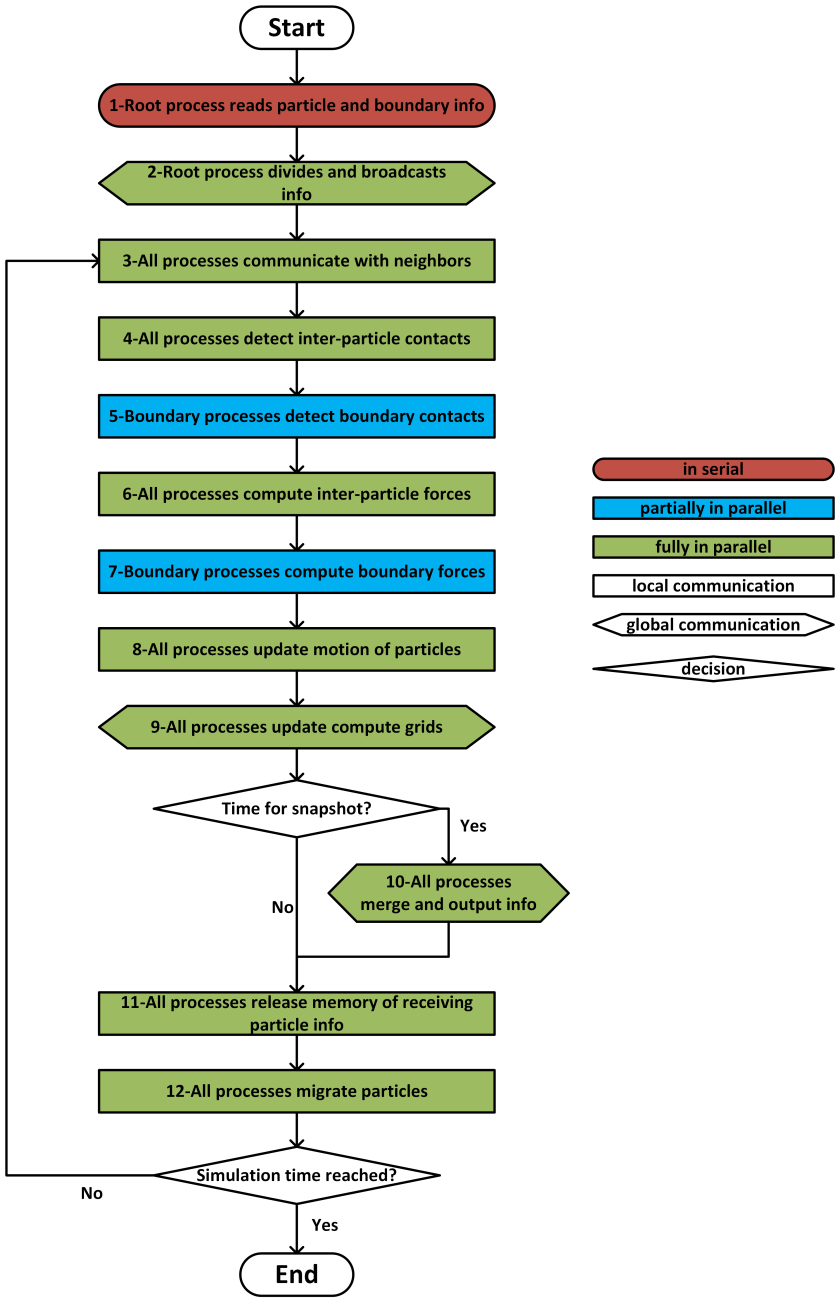


Figure 4: Flowchart of the parallel algorithm of 3D DEM.

Each block is mapped to and computed by an individual process, so there are four processes, p0 to p3. Each process needs to communicate with each other to determine its own boundary conditions. For example, process p3 needs to know those particles from process p1 that are enclosed by the pink rectangular box, those from process p2 enclosed by the red rectangular box, and those from process p0 enclosed by the green square box. A detailed movie records how those particles move across the borders and collide with particles from other blocks, and it reveals that each process is able to determine its boundary conditions accurately. The overall motion of the 162 particles through parallel computing is observed to be the same as that observed in serial computing.

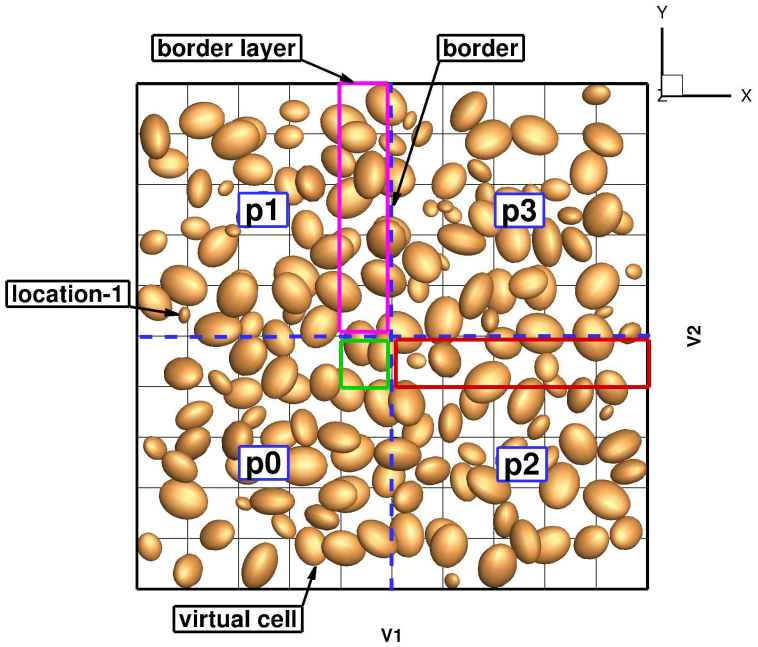


Figure 5: Illustration of interblock communication.

3.4. Other aspects in the parallelism

Many other aspects of the 3D DEM parallelism are detailed in the paper (Yan and Regueiro, 2018b,c), such as across-border migration, load balance and adaptive compute grids, minimizing global communication, OOP/C++, STL, Boost C++ libraries, avoiding large memory consumption with MPI,

etc. These details may have material influence on optimizing and improving the performance of the parallel 3D DEM code.

4. Theoretical iso-efficiency relation analysis of 3D DEM

In this section the general iso-efficiency relation for 3D DEM computation is derived and serves as a theoretical tool to analyze the memory usage feature in parallel computing. Note it is not used to study code structure and CPU usage features that are associated with complex particle shapes and geometric contact resolution process, which require much more computational cost than other components, as described in Section 2.4.

According to Eq. (1), the execution time of a parallel program executing on p processors is

$$T(n,p) = \sigma(n) + \varphi(n)/p + \kappa(n,p). \tag{8}$$

The serial program does not have interprocessor communication or synchronization, so the execution time is

$$T(n,1) = \sigma(n) + \varphi(n). \tag{9}$$

In order to sustain the same level of efficiency as the number of processors p increases, problem size n must increase to satisfy inequality (10),

$$T(n,1) \geq CT_o(n,p), \tag{10}$$

where $T_o(n,p) = (p-1)\sigma(n) + p\kappa(n,p)$, and $C = \varepsilon/(1-\varepsilon)$ is a constant related to efficiency ε. This is called the *iso-efficiency metric*, which is used to determine the scalability of a parallel system.

For 3D DEM, supposing n is the number of particles and p the number of processors used, we are able to derive the iso-efficiency metric following the analysis by Michael (2003). As mentioned in Section 2.4, in both serial and parallel computing of complex-shaped 3D DEM, the $O(n^2)$ neighbor search 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. The practical time complexity of 3D DEM falls between $O(n^2)$ and $O(n)$. Firstly we let

$$T(n,1) = O(n^2),$$

and the time needed to perform communications which are mostly through 2D surface layers is

$$\kappa(n,p) = \left(\frac{\sqrt[3]{n}}{\sqrt[3]{p}}\right)^2 = \left(\frac{n}{p}\right)^{\frac{2}{3}}.$$

Now Eq. (10) gives

$$n^2 \geq Cp\left(\frac{\sqrt[3]{n}}{\sqrt[3]{p}}\right)^2 \Longrightarrow n \geq Cp^{\frac{1}{4}}. \tag{11}$$

The amount of memory needed to represent a problem of size n is $M(n) = n$, then the scalability function, which indicates how the amount of memory used per processor must increase as a function of p in order to maintain the same level of efficiency, is calculated as follows,

$$\frac{M(f(p))}{p} = \frac{Cp^{\frac{1}{4}}}{p} = Cp^{-\frac{3}{4}}. \tag{12}$$

From Eq. (11), when the number of processors p increases from 1 to 10,000, the number of particles n only needs to increase 10 times for maintaining the same efficiency; this is easily satisfied in practical computations. Equation (12) states that when the number of processors increases, the memory requirement per processor decreases.

Secondly we let

$$T(n,1) = O(n),$$

and obtain

$$\frac{M(f(p))}{p} = \frac{Cp^{\frac{1}{4}}}{p} = C. \tag{13}$$

Equation (13) states that when the number of processors increases, the memory requirement per processor remains constant. In both cases, a parallel system of 3D DEM is perfectly scalable, according to Michael (2003): " A parallel system is perfectly scalable if the same level of efficiency can be sustained as the number of processors are increased by increasing the size of the problem being solved." Overall, the memory scalability function is a nonlinearly decreasing function of the number of processors, which is an intrinsic feature of 3D DEM.

5. DOD SUPERCOMPUTERS

The target architectures in this work are three of the DoD supercomputers: Spirit, Excalibur and Thunder. Spirit 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 has two Sandy Bridge-based Intel Xeon CPU E5-2670 2.60 GHz and 32 GB memory. The cluster of compute nodes are interconnected through 4x FDR InfiniBand network with enhanced LX hypercube topology.

Excalibur is a Cray XC40 System located at the ARL DSRC. It has 3,098 compute nodes each with 32 cores (99,136 total compute cores), 399 TBytes of memory, and is rated at 3.7 peak PFLOPS. Each compute node has two Haswell-based Intel Xeon CPU E5-2697 v3 2.60 GHz and 128 GB memory. The interconnect is Cray Aries with Dragonfly topology.

Thunder is an SGI ICE X System located at the AFRL DSRC. It has 3,216 standard compute nodes each with 36 cores (115,776 standard compute cores), 442.37 TBytes of memory, and is rated at 5.62 peak PFLOPS. Each compute node has two Haswell-based Intel Xeon CPU E5-2697 v3 2.60 GHz and 128 GB memory. The interconnect is 4x FDR InfiniBand with enhanced LX hypercube.

The Intel Xeon CPU E5-2670 was unveiled in 2012 as the fourth-generation eight-core architecture ("Sandy Bridge"), and some of the new and extended features in connection with performance improvement are:

- Four memory channels, and memory speed increases to 1600 MHz.
- Ring to connect on-chip L3 cache with cores, system agent, memory controller, and QPI agent and I/O controller to increase the scalability.
- L3 cache per core has been increased from 2 MB to 2.5 MB, and the total L3 cache size is 20 MB.
- INTEL Quickpath Interconnect (QPI) link rate increases to 8 GT/s and two QPI links to second socket.
- New AVX unit with wider vector registers of 256 bit.

The Sandy Bridge-based node of Spirit has two Xeon E5-2670 processors, each having eight cores. Each core has 64 KB of L1 cache (32 KB data and 32 KB instruction) and 256 KB of L2 cache. All eight cores share 20 MB of last level cache (LLC), which is also called L3 cache. Sandy Bridge has a bi-directional 32-byte ring interconnect that connects the 8 cores, the L3

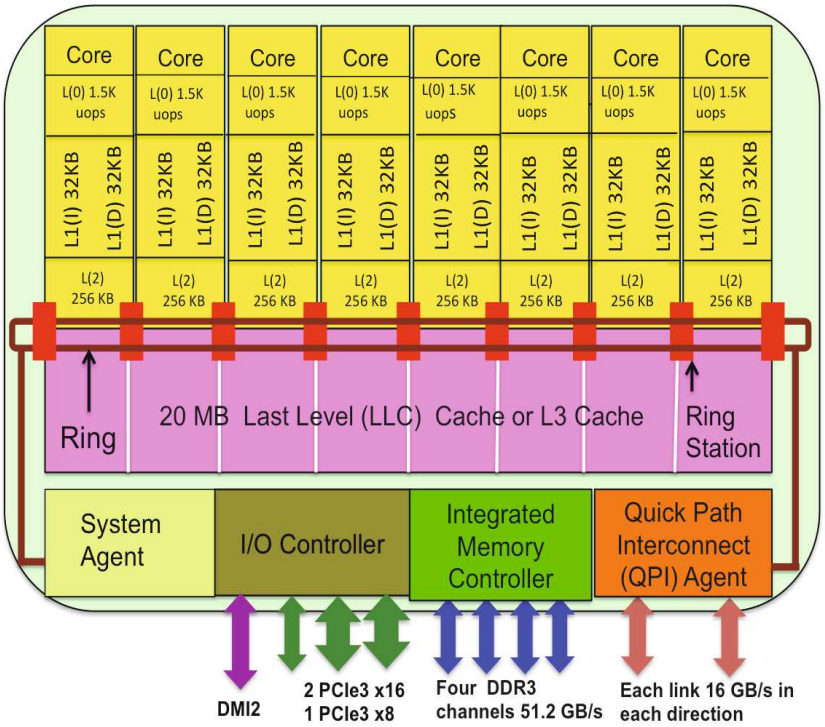


Figure 6: Schematic diagram of a Sandy Bridge processor (permission to use the figure granted by Saini) (Saini et al., 2013).

cache, the QPI agent and the integrated memory controller, as illustrated by Figure 6. The L3 cache is divided into eight slices/blocks, which are connected to the eight cores, and the system agent through a ring interconnect. Each core can address the entire cache, and each slice/block has a full cache pipeline.

6. PERFORMANCE ON THREE DOD SUPERCOMPUTERS

6.1. Types of DEM simulations

Overall, the problems that are modeled by 3D DEM fall into two main categories:

- static or quasi-static problems: laboratory tests such as oedometer compression, isotropic compression, conventional or true triaxial compression, in-situ Cone Penetration Test (CPT), static load test of cast-in-place piles, etc.

- dynamic problems: sand pluviation or deposition with gravity, collapse of particle assemblage, landslide under gravity, explosion beneath soil, installation of precast piles by means of hammers, sand dune movement, etc.

These two types of problems exhibit different features of particle interaction in 3D DEM simulations. The most pronounced difference is that particles come in and out of contact frequently in dynamic simulations, while there is less contact rearrangement in static or quasi-static modeling. In this work, static/quasi-static simulations of 500 iterations are carefully selected as our representative tests in evaluating parallel performance. There are three reasons for this selection:

1. static/quasi-static simulations allow an accurate control of constant time steps used in the explicit integration scheme.
2. our tests go across five orders of magnitude of simulation scale in terms of number of particles, namely, 2.5k, 12k, 150k, 1 million and 10 million;
3. for each simulation scale we attempt to obtain the run time performance employing from 1 core to 32,768 cores (2,048 nodes), thus both the problem size and the number of processors are investigated across a wide range.

6.2. Performance measurement on the three supercomputers

Speedup and efficiency data from Spirit, Excalibur and Thunder are populated in the following tables. Note that the node speedup/efficiency data are measured as well as the core speedup/efficiency for all of the simulation scales, except the 10 million particle simulation which cannot be accomplished using one core due to limit access to the supercomputers.

Firstly, it is observed that the core speedup is much higher than the node speedup across all of the simulation scales. For example, for 2.5k particles using 4 nodes the node speedup is 1.9 and core speedup is 19.5, for 150k particles using 16 nodes the node speedup is 29.9 and core speedup is 3,766.6, and for 1 million particles using 128 nodes the node speedup is 1,464.3 and core speedup is 167,882.4.

Secondly, for the smaller scale of 2.5k and 12k, as shown in Table 1, nearly all of the node efficiency and core efficiency are smaller than 1 (100%); however, for the larger scale of 150k, 1M and 10M, as shown in Table 2, both the node efficiency and core efficiency are greater than 1 (100%), and the core efficiency is much higher than the node efficiency.

Table 1: Speedup and efficiency of small scale simulations on Spirit

nodes	cores	time (s)	speedup	efficiency	speedup	efficiency
2.5k particles			node		core	
	1	129.41			1.00	1.00
1	16	12.82	1.00	1.00	10.10	0.63
2	32	9.53	1.35	0.67	13.58	0.42
3	48	7.70	1.66	0.55	16.80	0.35
4	64	6.63	1.93	0.48	19.51	0.30
6	96	5.50	2.33	0.39	23.52	0.25
8	128	4.92	2.60	0.33	26.30	0.21
16	256	3.69	3.47	0.22	35.03	0.14
32	512	3.35	3.82	0.12	38.57	0.08
64	1024	2.38	5.40	0.08	54.47	0.05
12k particles						
	1	1265.97			1.00	1.00
1	16	58.51	1.00	1.00	21.64	1.35
2	32	34.09	1.72	0.86	37.13	1.16
4	64	19.72	2.97	0.74	64.20	1.00
6	96	14.68	3.99	0.66	86.26	0.90
8	128	12.29	4.76	0.60	103.03	0.80
16	256	8.73	6.70	0.42	144.94	0.57
32	512	6.11	9.58	0.30	207.37	0.41
64	1024	5.31	11.01	0.17	238.30	0.23

Table 2: Speedup and efficiency of large scale simulations on Spirit

nodes	cores	time (s)	speedup	efficiency	speedup	efficiency
150k particles			node		core	
	1	210316.10			1.00	1.00
1	16	1671.39	1.00	1.00	125.83	7.86
2	32	541.27	3.09	1.54	388.56	12.14
4	64	221.77	7.54	1.88	948.33	14.82
8	128	105.84	15.79	1.97	1987.19	15.52
16	256	55.84	29.93	1.87	3766.59	14.71
32	512	33.40	50.04	1.56	6296.40	12.30
64	1024	22.39	74.63	1.17	9391.48	9.17
128	2048	18.12	92.25	0.72	11607.72	5.67
1M particles						
	1	13860166.45			1.00	1.00
1	16	120887.41	1.00	1.00	114.65	7.17
2	32	28095.91	4.30	2.15	493.32	15.42
4	64	5747.83	21.03	5.26	2411.37	37.68
8	128	1516.96	79.69	9.96	9136.83	71.38
16	256	462.91	261.15	16.32	29941.42	116.96
32	512	212.85	567.96	17.75	65118.53	127.18
64	1024	121.83	992.22	15.50	113762.03	111.10
128	2048	82.56	1464.26	11.44	167882.44	81.97
256	4096	61.70	1959.16	7.65	224624.96	54.84
512	8192	56.00	2158.71	4.22	247503.99	30.21
10M particles						
1	16	20269795.77	1.00	1.00		
2	32	300396.78	67.48	33.74		
4	64	76511.85	264.92	66.23		
8	128	28266.40	717.10	89.64		
16	256	10568.23	1917.99	119.87		
32	512	7288.38	2781.11	86.91		
64	1024	6640.23	3052.57	47.70		
128	2048	6548.81	3095.19	24.18		
256	4096	6552.20	3093.59	12.08		
512	8192	6604.70	3069.00	5.99		
768	12288	7136.70	2840.22	3.70		
1024	16384	8382.69	2418.05	2.36		

Table 3: Speedup and efficiency of small scale simulations on Excalibur

nodes	cores	time (s)	speedup	efficiency	speedup	efficiency
2.5k particles			node		core	
	1	122.83			1.00	1.00
1	32	6.82	1.00	1.00	18.01	0.56
2	64	4.87	1.40	0.70	25.21	0.39
3	96	4.04	1.69	0.56	30.43	0.32
4	128	3.55	1.92	0.48	34.61	0.27
6	192	2.90	2.35	0.39	42.40	0.22
8	256	2.83	2.41	0.30	43.40	0.17
16	512	2.08	3.27	0.20	58.93	0.12
32	1024	1.92	3.55	0.11	63.90	0.06
64	2048	3.08	2.21	0.03	39.87	0.02
12k particles						
	1	1660.74			1.00	1.00
1	32	28.75	1.00	1.00	57.77	1.81
2	64	16.35	1.76	0.88	101.59	1.59
4	128	9.53	3.02	0.75	174.26	1.36
6	192	7.42	3.88	0.65	223.96	1.17
8	256	7.53	3.82	0.48	220.65	0.86
16	512	4.45	6.46	0.40	373.27	0.73
32	1024	8.10	3.55	0.11	204.91	0.20
64	2048	4.68	6.15	0.10	355.04	0.17

Table 4: Speedup and efficiency of large scale simulations on Excalibur

nodes	cores	time (s)	speedup	efficiency	speedup	efficiency
150k particles			node		core	
	1	199818.73			1.00	1.00
1	32	673.98	1.00	1.00	296.48	9.26
2	64	254.91	2.64	1.32	783.89	12.25
4	128	108.30	6.22	1.56	1845.09	14.41
8	256	51.78	13.02	1.63	3858.80	15.07
16	512	28.74	23.45	1.47	6952.35	13.58
32	1024	21.05	32.02	1.00	9492.57	9.27
64	2048	15.16	44.45	0.69	13178.78	6.43
128	4096	19.25	35.01	0.27	10378.49	2.53
1M particles						
	1	15182678.83			1.00	1.00
1	32	32652.50	1.00	1.00	464.98	14.53
2	64	6357.97	5.14	2.57	2387.98	37.31
4	128	1678.76	19.45	4.86	9044.01	70.66
8	256	561.95	58.11	7.26	27017.74	105.54
16	512	217.47	150.15	9.38	69815.96	136.36
32	1024	117.18	278.65	8.71	129567.39	126.53
64	2048	74.29	439.53	6.87	204373.15	99.79
128	4096	78.35	416.76	3.26	193785.30	47.31
256	8192	110.93	294.34	1.15	136863.02	16.71
512	16384	155.88	209.48	0.41	97401.82	5.94
10M particles						
1	32	5912563.15	1.00	1.00		
2	64	93400.65	63.30	31.65		
4	128	19859.66	297.72	74.43		
8	256	118101.16	50.06	6.26		
16	512	5562.87	1062.86	66.43		
32	1024	21245.00	278.30	8.70		
64	2048	19058.43	310.23	4.85		
128	4096	19684.92	300.36	2.35		
256	8192	22038.31	268.29	1.05		
512	16384	26321.64	224.63	0.44		
768	24576	32998.37	179.18	0.23		
1024	32768	43745.79	135.16	0.13		

Table 5: Speedup and efficiency of small scale simulations on Thunder

nodes	cores	time (s)	speedup	efficiency	speedup	efficiency
2.5k particles			node		core	
	1	105.25			1.00	1.00
1	36	7.44	1.00	1.00	14.15	0.39
2	72	4.79	1.55	0.78	21.99	0.31
3	108	3.78	1.97	0.66	27.86	0.26
4	144	3.47	2.14	0.54	30.30	0.21
6	216	3.00	2.48	0.41	35.14	0.16
8	288	2.82	2.63	0.33	37.28	0.13
16	576	2.27	3.28	0.20	46.39	0.08
32	1152	2.34	3.17	0.10	44.88	0.04
64	2304	3.50	2.13	0.03	30.08	0.01
12k particles						
	1	1146.04			1.00	1.00
1	36	23.86	1.00	1.00	48.04	1.33
2	72	13.60	1.75	0.88	84.24	1.17
4	144	9.28	2.57	0.64	123.52	0.86
6	216	6.95	3.43	0.57	164.88	0.76
8	288	7.97	2.99	0.37	143.76	0.50
16	576	4.50	5.31	0.33	254.92	0.44
32	1152	8.04	2.97	0.09	142.55	0.12
64	2304	7.02	3.40	0.05	163.19	0.07

The phenomenon that core efficiency goes appreciably higher than node efficiency is interpreted as follows: extending from 1 core to 16 cores (1 node) delivers an intra-node MPI performance gain, but increasing from 1 node to multi-nodes acquires an inter-node (more accurately, a mixture of inter-node and intra-node) MPI performance gain. The inter-node mode gains less performance improvement than the pure intra-node mode because of its higher MPI transmission overhead, thus leading to lower speedup and efficiency.

Hereby the node speedup/efficiency is chosen for performance study for two reasons: (1) to be conservative on the measurement of speedup and efficiency, (2) in light of the large computational resources (compute nodes) used in the simulations.

6.3. Strong scaling view of the speedup and efficiency

Figure 7 plots the speedup and efficiency for static simulations at the five different scales of number of particles. The speedup exhibits a monotonically increasing relationship with respect to the number of compute nodes at all scales. The efficiency exhibits a monotonically decreasing trend related to the number of compute nodes for 2.5k and 12k, and the values of efficiency are below 1, but it captures a peak value on the scale of 150k, 1M and 10M particles and reveals values higher than 1, namely, superlinear speedup. The superlinear speedup is particularly pronounced; for example, the efficiency goes as high as 1.97 (197%) at 8 nodes in the 150k test; and 17.75 (1,775%)

Table 6: Speedup and efficiency of large scale simulations on Thunder

nodes	cores	time (s)	speedup	efficiency	speedup	efficiency
150k particles			node		core	
	1	210316.10			1.00	1.00
1	36	1671.39	1.00	1.00	439.28	12.20
2	72	541.27	2.40	1.20	1056.40	14.67
4	144	221.77	5.05	1.26	2216.88	15.39
8	288	105.84	9.87	1.23	4335.82	15.05
16	576	55.84	15.64	0.98	6868.96	11.93
32	1152	33.40	20.80	0.65	9134.83	7.93
64	2304	22.39	25.01	0.39	10987.98	4.77
128	4608	18.12	22.58	0.18	9920.53	2.15
1M particles						
	1	13860166.45			1.00	1.00
1	36	120887.41	1.00	1.00	400.83	11.13
2	72	28095.91	4.31	2.15	1727.14	23.99
4	144	5747.83	22.48	5.62	9009.22	62.56
8	288	1516.96	72.99	9.12	29256.06	101.58
16	576	462.91	153.67	9.60	61594.81	106.94
32	1152	212.85	267.41	8.36	107185.75	93.04
64	2304	121.83	371.52	5.81	148916.20	64.63
128	4608	82.56	465.16	3.63	186447.45	40.46
256	9216	61.70	438.07	1.71	175590.66	19.05
512	18432	56.00	311.99	0.61	125053.99	6.78
10M particles						
1	36	20269795.77	1.00	1.00		
2	72	300396.78	59.33	29.66		
4	144	76511.85	206.16	51.54		
8	288	28266.40	563.05	70.38		
16	576	10568.23	807.73	50.48		
32	1152	7288.38	856.69	26.77		
64	2304	6640.23	858.00	13.41		
128	4608	6548.81	864.82	6.76		
256	9216	6552.20	859.53	3.36		
512	18432	6604.70	832.83	1.63		
768	27648	7136.70	512.89	0.67		
1024	36864	8382.69	374.96	0.37		

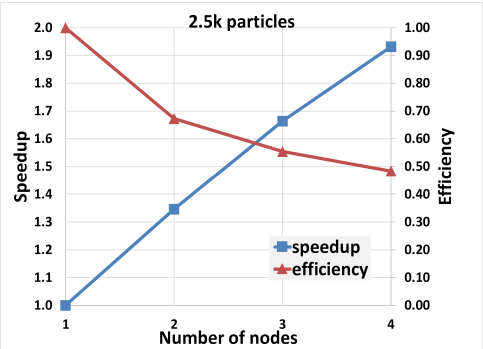
at 32 nodes, and 7.65 (765%) at 256 nodes in the 1 million particle test.

It should be noted that for all of the 1-node tests across the five scales, the memory size is sufficiently adequate to satisfy the computation and does not cause swap-out to hard drive. For example, the 1 million particle test requires approximately 3GB memory on the 1-node test, which is significantly below the 32GB node memory provided by the DoD Spirit supercomputer.

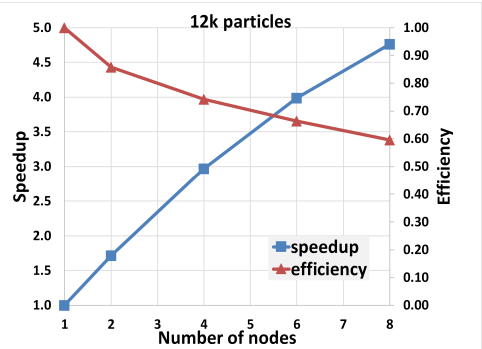
6.4. Integrated view of the speedup and efficiency

Figure 8 compiles all of the speedup and efficiency data from static simulations on Spirit at the five different scales. A loglog graph is plotted due to the wide range of problem size and number of processors. It is observed that speedup is an increasing function of the problem size for any fixed number of processors.

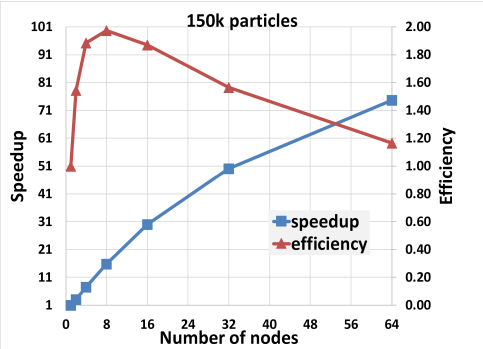
It is clearly seen from Figure 8(b) that the efficiencies of 150k, 1 million and 10 million particle simulations go beyond 1 (100%), and the larger the



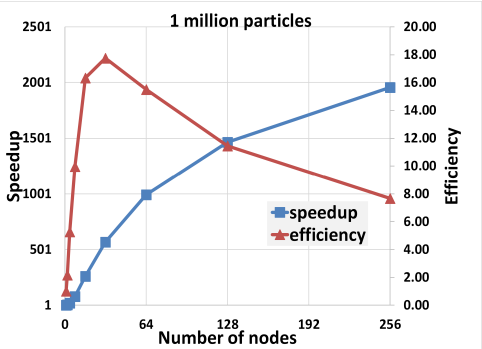
(a) 2.5k particles.



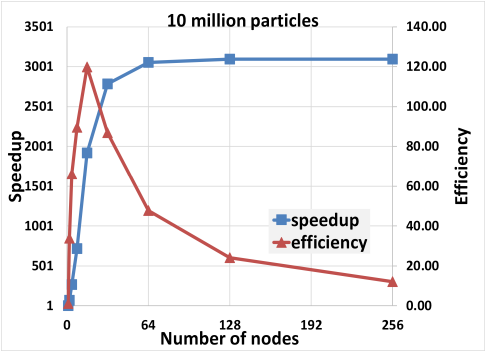
(b) 12k particles.



(c) 150k particles.



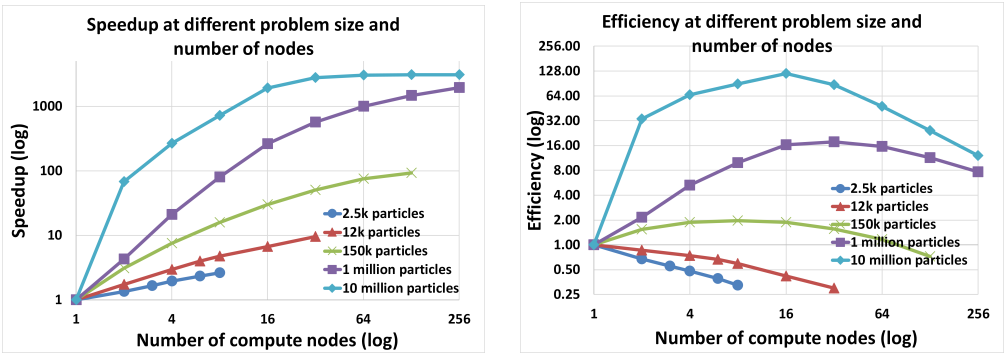
(d) 1 million particles.



(e) 10 million particles.

Figure 7: Speedup and efficiency across orders of magnitude of simulation scale in terms of number of particles.

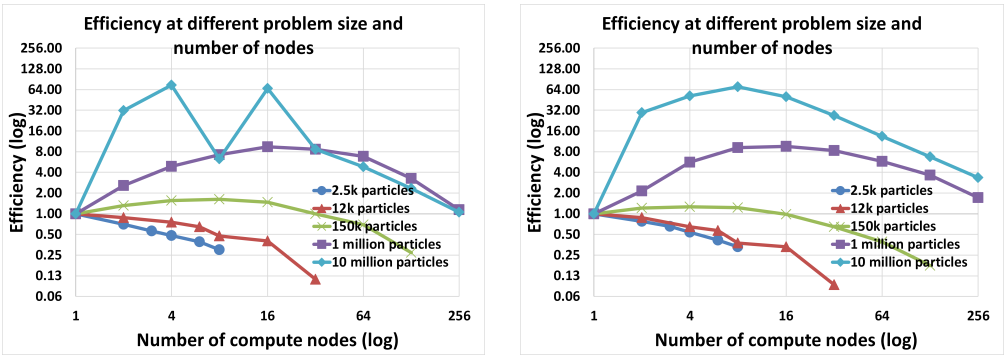
scale, the stronger is the superlinear speedup. The same phenomenon is replicated on the Excalibur and Thunder supercomputers, as shown in Figure 9,



(a) Speedup. (b) Efficiency.

Figure 8: Speedup and efficiency across orders of magnitude simulation scale on Spirit.

whereby only the 10 million test on Excalibur exhibits a bit fluctuation using 8 nodes (this replicable fluctuation on Excalibur is most perplexing and we cannot explain it).



(a) On Excalibur. (b) On Thunder.

Figure 9: Efficiency on Excalibur and Thunder.

Although strong superlinear speedups are observed for larger scale simulation of 150k, 1M and 10M particles on all of the three supercomputers, the efficiencies are not monotonically increasing: increase first but decrease later. This is worth further investigation.

7. CAUSES OF SUPERLINEAR SPEEDUP IN 3D DEM

There are quite a few sources that contribute to the superlinear speedup in parallel computing as summarized in section 1.3: hidden memory latency,

subdivision of system overhead, randomized algorithms, tiered memory architecture, shift in time fraction spent on different-speed tasks, load imbalance, serial algorithm steps, interprocessor communication, etc.

Gustafson (1990) also stated that the decrease in uniprocessor performance appears as another source of parallel inefficiency as a problem is "spread out" among processors, and Sandia experiments (Gustafson, 1990) showed the uniprocessor performance is an increasing but asymptotic function of problem size.

In studying the tiered memory and superlinear speedup, Gustafson (1990) mentioned the subtle effect when a processor has a data cache, and the regime is entered in a manner transparent to the programmer. He believed that the reasoning on superlinear speedup is applicable to both shared memory computers and distributed memory machines. He made the following statement: *"It is not anomalous. It is as inescapable as the laws of physics that processor-memory speeds increase as a problem is spread out over many processors."* and *"Superlinear parallel speedup, far from being the result of 'inefficient' serial execution, becomes inescapable when the performance model makes realistic assumptions about the speed of memory and the way problems scale."*

Among all of these possible sources, we will focus on four of them: (1) memory consumption; (2) interprocess MPI communication; (3) tiered memory hierarchy; (4) uniprocessor performance, because other sources such as load imbalance has been minimized in our 3D DEM, referring to Yan and Regueiro (2018b,c).

Yan and Regueiro (2018c) pointed out that the interprocess MPI communication time is a decreasing function of the number of compute nodes, $O((\log p)/p)$, whereas the parallel overhead percentage (ratio of MPI communication time to total execution time) is an increasing function, $O(\log p)$, and is around 10% with granularity-optimized p for complex-shaped 3D DEM, where p denotes the number of compute nodes. That means the 2D MPI communication decreases slower than the 3D computation with respect to increasing number of compute nodes, therefore MPI communication change with p hinders rather than facilitates the parallel speedup. Furthermore, it is impractical to differentiate the MPI-associated memory usage from the computation-associated memory usage in an OS process, so we leave the process-consumed memory as a whole.

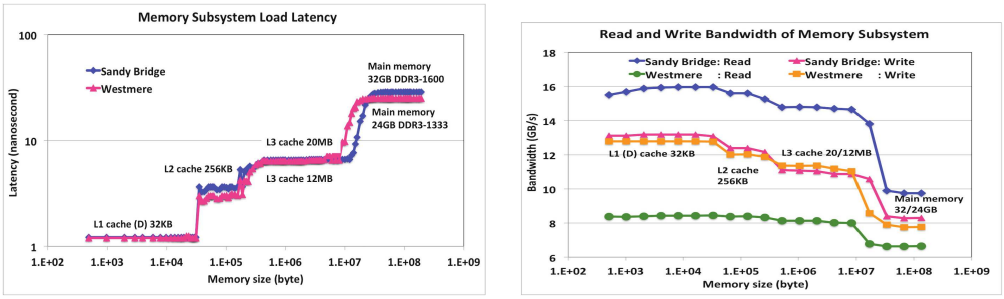
The DEM features high demand on CPU clock rate (frequency) but exerts low demand on memory usage (so called high-CPU-low-memory). In

Yan and Regueiro (2018c), it has been theoretically proved that memory requirement per processor decreases when number of processors and problem size increase to maintain the same efficiency (iso-efficiency) for 3D DEM. Therefore it is not surprising that the larger the simulation scale in terms of number of particles, the more pronounced superlinear speedup may occur, because it becomes more likely that the process-partitioned data fit into CPU caches.We will use replicable experiment data measured through advanced performance monitor tools to justify this conjecture.

8. MEASUREMENTS ON THREE SUPERCOMPUTERS

8.1. Cache latency and bandwidth

Saini et al. (2013) measured the memory latency and bandwidth of Sandy Bridge processors, as shown in Figure 10. The latency exhibits a step function pattern: L1 cache latency is 1.2 ns; L2 cache latency is 3.5 ns; L3 cache latency is 6.5 ns; and main memory latency is 28 ns, which is approximately 4.3 times as high as the L3 cache latency.



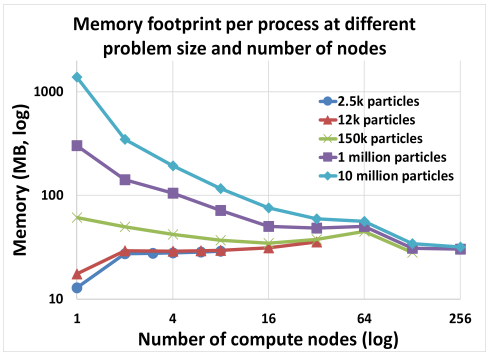
(a) Latency. (b) Bandwidth.

Figure 10: Memory latency and bandwidth of Sandy Bridge processors (permission to use the figure granted by Saini) (Saini et al., 2013).

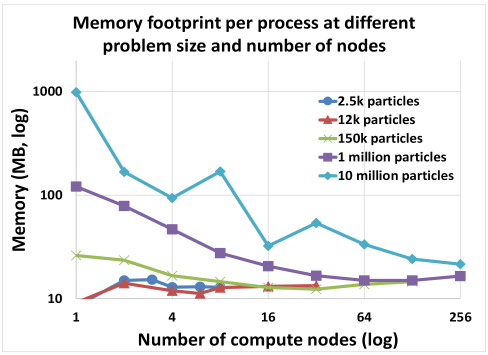
8.2. Memory footprint

To measure the memory consumption, the system function `getrusage` is called in the code to acquire the resident memory per process. Each process prints out its own memory footprint and an average value is then calculated.

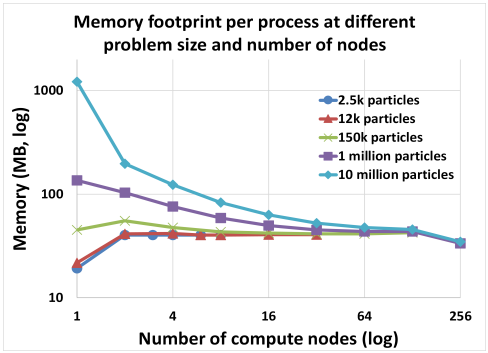
Figure 11 plots the memory consumption per process across five orders of magnitude of simulation scale on Spirit, Excalibur and Thunder, respectively. Overall, the memory footprints of 150k, 1M and 10M particles decrease with



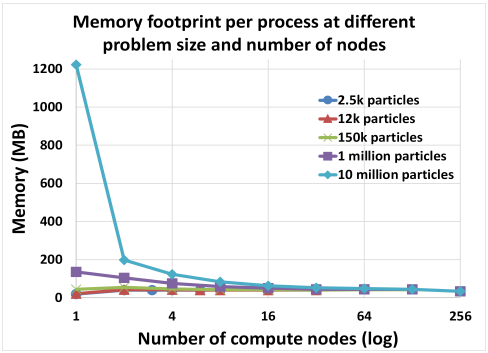
(a) Spirit (log-log).



(b) Excalibur (log-log).



(c) Thunder (log-log).



(d) Thunder (log).

Figure 11: Memory consumption per process/core on the three DoD supercomputers.

the number of nodes, whereas for 2.5k and 12k particles, they either increase slightly on Spirit or stay nearly constant (actually decrease very slightly) on Excalibur and Thunder. Figure 11(d) depicts memory in linear scale on Thunder so that we can see clearly that the overall memory consumption is below 150 MB per process/core except the case of 10 million particles using 1 compute node. The memory consumption per core is low in the sense of scientific computing. However, recalling that the L3 cache per core is 2.5 MB on Sandy bridge-based CPU, the memory consumption per core is still one to two orders of magnitude larger than the L3 cache size.

It is particularly important to observe that: for 2.5k and 12k particles the memory consumption per process does not decrease, and it corresponds to no superlinear speedup; whereas for 150k, 1M and 10M particles the memory footprint per process does decrease, and it corresponds to the superlinear

speedup. The change of memory footprint per process plays a critical role in determining speedup characteristics.

That the memory consumption per process decreases with an increasing number of nodes in large-scale simulations but does not in small-scale ones, depends on the inherent code design and underlying data structures: the memory allocation for particles and interparticle contacts dominates other components such as eigenvalue solvers only when there are adequately large number of particles.

Referring to Eq. (12) in Section 4, which states that the memory scalability function is a nonlinearly decreasing function of the number of processors, an intrinsic feature of 3D DEM, it is clear that the memory footprint measurements agree well with the theoretical analysis.

8.3. PAPI Implementation

In order to measure cache misses or other performance data, the Performance Application Programming Interface (PAPI) is used for programming the performance measurement part in the parallel code. Referring to `http://icl.cs.utk.edu/papi/`, PAPI provides programmers with a consistent interface and methodology for use of the performance counter hardware and processor events found in most major microprocessors, and a few example are listed in Table 7.

preset events	description
PAPI_L1_TCM	L1 cache misses
PAPI_L2_TCM	L2 cache misses
PAPI_L3_TCM	L3 cache misses
PAPI_L3_ICR	L3 instruction cache reads
PAPI_L3_ICW	L3 instruction cache writes
PAPI_FP_INS	Floating point instructions
PAPI_FP_OPS	Floating point operations
PAPI_MEM_SCY	Cycles Stalled Waiting for memory accesses

Table 7: Example of PAPI preset events.

According to `https://icl.cs.utk.edu/projects/papi/wiki/PAPITopics:SandyFlops`, PAPI only guarantees 3 programmable counters at a given time. That means we will have to re-run the same simulation if we intend to measure more than 3 events. In addition, Intel has disabled the floating point

counters in the Haswell-based CPU architecture as of today, however Sandy Bridge and Ivy Bridge products are allows to measure FLOPS.

8.4. Cache miss per second

Note that PAPI_LX_TCM (X=1, 2, 3) gives the total cache misses (TCM), not the TCM rate. In order to study and compare performance using a consistent standard across different simulation scales, TCM rate is employed by dividing TCM by execution time. Tables 8 and 9 list the L1, L2 and L3 cache misses and execution time for simulations of 2.5k, 12k particles, and simulations of 150k, 1M, 10M particles measured on Thunder, respectively. Figure 12 plots the total cache miss (TCM) per second of L1, L2 and L3 caches across all of the five orders: 2.5k, 12k, 150k, 1M and 10M particles on Thunder.

Now it can be seen that for 2.5k and 12k particles, even though the memory usage per process decreases nearly unnoticeably as shown in Figure 11(c), the TCM per second tends to give a clearly decreasing trend, revealed in Figure 12(a) and (b). This indicates the sensitivity of TCM to the change of memory consumption.

The situations for 150k, 1M and 10M particles deliver different details from that of 2.5k and 12k particles, even though they also exhibit a decreasing trend of TCM rate. In the latter, the TCM rates of L1, L2 and L3 caches decrease by nearly the same order of magnitude. However, in the former the three cache level TCM rates decrease differently: L3_TCM rate $\gg$ L2_TCM rate $\gg$ L1_TCM rate. For example, L3_TCM rate decreases from 1.8E+7 to 6.5E+4 (nearly three orders of magnitude), L2_TCM rate decreases from 8.1E+7 to 1.4E+6 (one order of magnitude), and L1_TCM rate decreases from 7.1E+6 to 3.0E+6 (by 50%) for 1 million particles.

In connection with the speedup shown in Figure 8 and 9, wherein 2.5k and 12k particles show no superlinear speedup whereas 150k, 1M and 10M particles exhibit strong superlinear speedup, we can conclude that the L3 cache contributes most significantly to the strong superlinear speedup among all of the three cache levels.

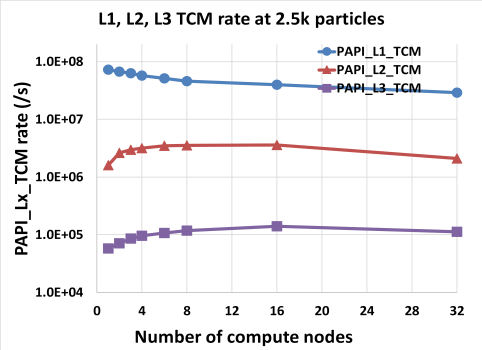
Figure 13 compiles various simulation scales for L3, L2 and L1 cache, respectively. It is seen that for each cache level, the 150k, 1M and 10M particle simulations have greater decrease of TCM rate than that of 2.5k and 12k particles.

Table 8: Cache misses of small scale simulations on Thunder

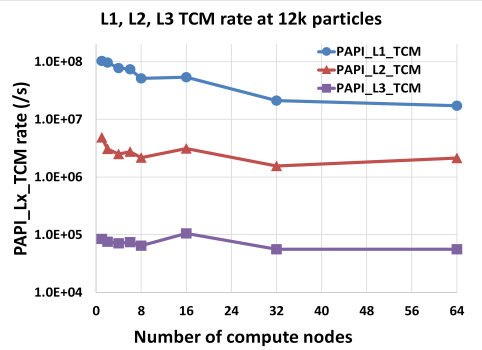
nodes	cores	PAPI_L3_TCM	PAPI_L2_TCM	PAPI_L1_TCM	time (s)
2.5k particles					
	1	1.7E+04	6.4E+09	1.9E+10	1.1E+02
1	36	4.3E+05	1.2E+07	5.4E+08	7.4E+00
2	72	3.4E+05	1.3E+07	3.2E+08	4.8E+00
3	108	3.3E+05	1.1E+07	2.4E+08	3.8E+00
4	144	3.4E+05	1.1E+07	2.0E+08	3.5E+00
6	216	3.2E+05	1.0E+07	1.5E+08	3.0E+00
8	288	3.3E+05	1.0E+07	1.3E+08	2.8E+00
16	576	3.2E+05	8.2E+06	9.0E+07	2.3E+00
32	1152	2.6E+05	4.9E+06	6.8E+07	2.3E+00
64	2304	2.2E+05	1.4E+07	6.9E+07	3.5E+00
12k particles					
	1	2.9E+07	2.0E+11	2.0E+11	1.1E+03
1	36	2.0E+06	1.2E+08	2.5E+09	2.4E+01
2	72	1.0E+06	4.2E+07	1.3E+09	1.4E+01
4	144	6.6E+05	2.3E+07	7.2E+08	9.3E+00
6	216	5.2E+05	1.9E+07	5.1E+08	7.0E+00
8	288	5.2E+05	1.7E+07	4.1E+08	8.0E+00
16	576	4.7E+05	1.4E+07	2.4E+08	4.5E+00
32	1152	4.5E+05	1.2E+07	1.7E+08	8.0E+00
64	2304	3.9E+05	1.5E+07	1.2E+08	7.0E+00

Table 9: Cache misses of large scale simulations on Thunder

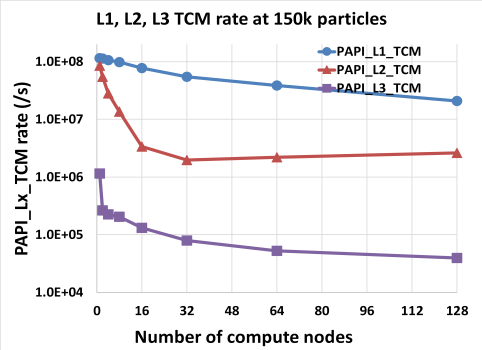
nodes	cores	PAPI_L3_TCM	PAPI_L2_TCM	PAPI_L1_TCM	time (s)
150k particles					
	1	9.0E+12	3.2E+13	2.2E+13	1.9E+05
1	36	4.9E+08	3.6E+10	4.8E+10	4.2E+02
2	72	4.6E+07	9.6E+09	2.0E+10	1.8E+02
4	144	1.9E+07	2.4E+09	8.9E+09	8.3E+01
8	288	8.8E+06	5.8E+08	4.2E+09	4.3E+01
16	576	3.5E+06	9.0E+07	2.1E+09	2.7E+01
32	1152	1.6E+06	4.0E+07	1.1E+09	2.0E+01
64	2304	8.9E+05	3.7E+07	6.5E+08	1.7E+01
128	4608	7.4E+05	4.9E+07	3.9E+08	1.9E+01
1M particles					
	1	9.5E+12	1.3E+13	7.5E+12	1.1E+07
1	36	5.0E+11	1.4E+12	9.9E+11	2.8E+04
2	72	8.1E+10	3.7E+11	3.0E+11	6.5E+03
4	144	6.8E+09	1.0E+11	1.0E+11	1.2E+03
8	288	3.8E+08	2.7E+10	3.9E+10	3.8E+02
16	576	4.0E+07	6.9E+09	1.6E+10	1.8E+02
32	1152	1.8E+07	1.9E+09	7.4E+09	1.0E+02
64	2304	8.5E+06	4.3E+08	3.5E+09	7.5E+01
128	4608	3.9E+06	8.2E+07	1.8E+09	6.0E+01
256	9216	1.7E+06	5.5E+07	1.1E+09	6.4E+01
512	18432	1.4E+07	1.5E+08	7.6E+08	9.0E+01
10M particles					
1	36	1.1E+12	1.5E+12	8.9E+11	5.4E+06
2	72	1.6E+10	2.7E+10	1.8E+10	9.0E+04
4	144	3.3E+09	7.8E+09	5.5E+09	2.6E+04
8	288	3.6E+08	2.0E+09	1.8E+09	9.5E+03
16	576	2.8E+07	5.7E+08	6.6E+08	6.6E+03
32	1152	2.3E+06	1.7E+08	4.7E+08	6.3E+03
64	2304	4.4E+05	9.6E+07	2.7E+08	6.3E+03
128	4608	2.5E+05	1.3E+08	1.8E+08	6.2E+03
256	9216	2.4E+05	1.2E+08	1.5E+08	6.2E+03
512	18432	8.7E+06	1.0E+08	1.2E+08	6.4E+03
768	27648	2.7E+07	1.5E+08	1.6E+08	1.0E+04
1024	36864	7.7E+07	2.2E+08	1.1E+08	1.4E+04



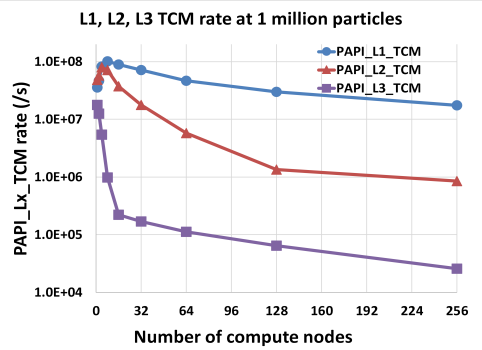
(a) 2.5k particles.



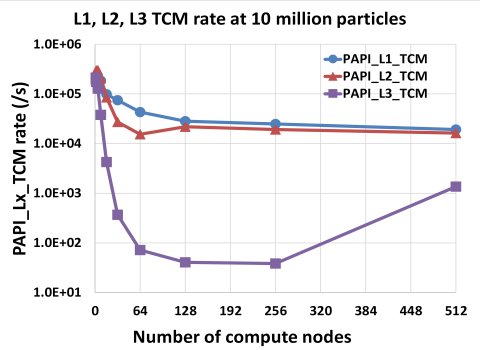
(b) 12k particles.



(c) 150k particles.

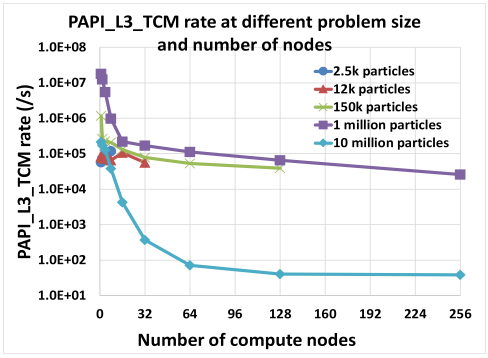


(d) 1M particles.

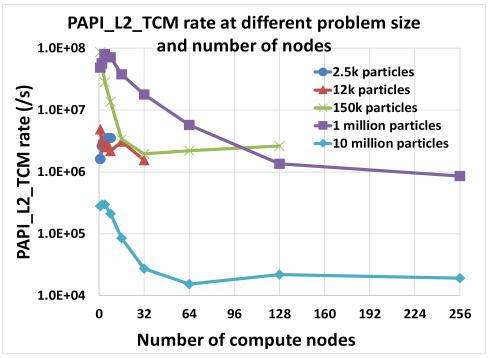


(e) 10M particles.

Figure 12: L1, L2 and L3 cache miss (per second) comparison across simulation scales on Thunder.

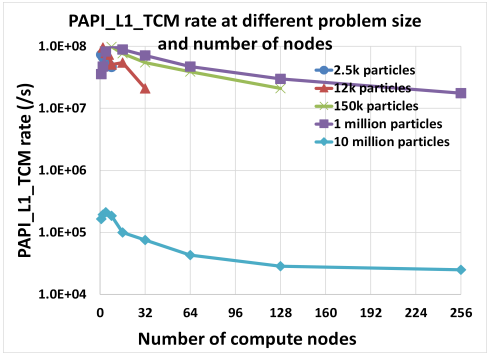


(a) L3 cache.



(b) L2 cache.

[



(c) L1 cache.

Figure 13: Cache miss (per second) measurement across orders of magnitude of simulation scale on Thunder.

8.5. CPU stall time

It is of great interest to investigate the CPU stall time (also called memory stall time) for all of the simulation scales and computational granularities. However, it turns out that the PAPI_MEM_SCY event (Cycles Stalled Waiting for memory accesses) does not support the latest Intel Sandybrige, Ivybridge or Haswell-based Xeon processors on DoD supercomputers, nor can the PAPI-based tools such TAU, PerfSuite, PCTookkit, etc, make such measurement. The Intel Performance Counter Monitor (PCM), which has been discontinued but leaves a fork at github.com, is used together with the Linux Perf tool for measurements. Because its installation and use require root privilege to modify OS system files (not allowed on DoD supercomputers), the Intel PCM can only be employed in a Dell T7500 Precision Linux workstation of dual hexa-core Intel Xeon X5690 Westmere-EP processors.

Table 10: CPU stall time measurement

	Intel PCM					Perf	
Particles	IPC	L3MISS (M)	L2MISS (M)	L3HIT	L2HIT	stallled cycles	cache miss
2.5k	1.46	5	151	0.96	0.80	21.06%	3.69%
12k	1.37	10	684	0.98	0.76	21.41%	1.65%
150k	0.80	367	9382	0.96	0.03	28.13%	15.41%
1M	0.38	1165	1180	0.01	0.01	55.65%	98.51%
10M	0.31	1208	1216	0.01	0	58.96%	99.19%

Table 10 lists the 1-node measurements from Intel PCM and Linux Perf across five orders of magnitude of simulation scale. As the simulation scale grows, the Instructions Per Cycle (IPC) decreases, the L3 cache misses (L3MISS) increases, the L3 cache hit rate (L3HIT) decreases, CPU stall cycles ratio increases, and the overall cache miss rate increases. In particular, the IPC drops from 1.46 to 0.38, and the CPU stall cycles ratio increases from 21% to 56%, when the number of particles grows from 2.5k to 1M.

From the following equations,

$$\begin{aligned} \text{CPU time} &= (\text{CPU exec clock cycles} + \text{memory stall cycles}) \times \text{cycle time}, \\ \text{stall cycles ratio} &= \frac{\text{memory stall cycles}}{\text{CPU exec clock cycles}}, \end{aligned} \tag{14}$$

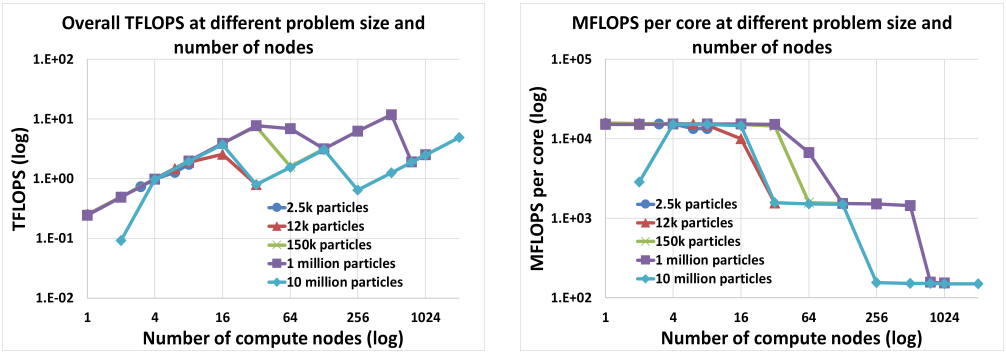
we gain insight into the CPU stall level. The computational granularity (CG) of 2.5k particles on 12-cores is approximately the optimal CG in parallel

computation of complex-shaped 3D DEM (Yan and Regueiro, 2018c). At this CG level, the CPU stall ratio is as low as approximately 21%, and the cache miss is as low as 3.7%, therefore the complex-shaped 3D DEM is characterized by CPU-bound rather than memory-bound, in addition to its "high-CPU-low-memory" feature.

Beware the CPU stall ratio is not the CPU usage (percentage of CPU used by a process) shown by "top" command, which is always 100% for each process in parallel computing of complex-shaped 3D DEM.

8.6. Uniprocessor performance and generalized speedup

As floating point counters have been disabled in the Haswell-based CPU architecture as of today, we are unable to measure the FLOPS on Thunder or Excalibur. Figure 14 compiles the FLOPS performance across five orders of magnitude of simulation scale (number of particles) on Spirit. Overall, the FLOPS performance increases when the number of compute nodes increases, whereas the FLOPS per core (uniprocessor performance) decreases with regard to increasing number of compute nodes.

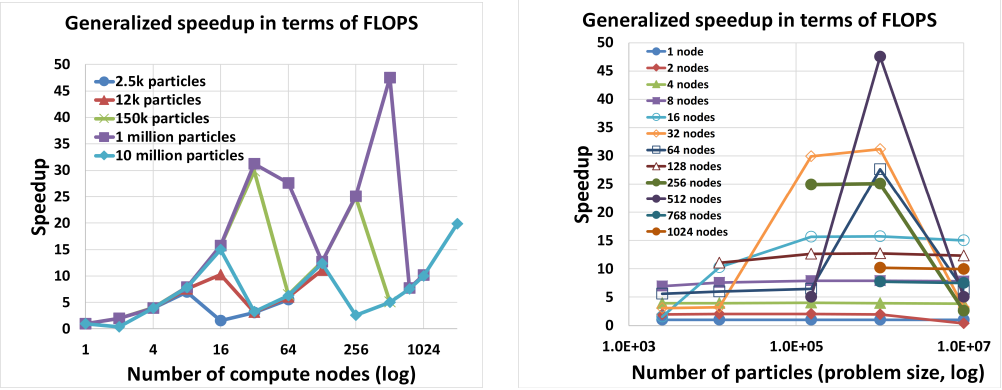


(a) Overall FLOPS. (b) Per core FLOPS.

Figure 14: Floating-point operation performance w.r.t number of compute nodes on Spirit (strong scaling).

Figure 15(a) presents the generalized speedups in terms of PAPI-measured overall FLOPS according to Eq. (6). Note the generalized speedups are calculated relative to the sequential measurement of 2.5k particles on a single node, which is a typical computational granularity (CG). Actually the TFLOPS's on a single node are 0.249, 0.251, 0.249, 0.243, 0.243 for 2.5k, 12k, 150k, 1M, 10M particles, respectively. They are so close that it makes little difference to select which one in evaluating the generalized speedup.

The generalized speedup curves (each corresponding to a fixed-size problem) with regard to number of compute nodes, as shown in Figure 15, exhibit an overall increasing trend, yet do not vary as smoothly as the speedup in terms of Eq. (1), as shown in Figure 8. As stated in Sun and Gustafson (1991), speedup is the restriction of generalized speedup to fixed work, therefore the gap between the generalized speedup and "conventional" speedup indicates other factors such as memory hierarchy have influence. Figure 15(b) plots the generalized speedup curves with regard to problem size, whereby it is observed the generalized speedup increases with an increasing problem size (number of particles).



(a) W.R.T number of compute nodes.

(b) W.R.T number of particles.

Figure 15: Generalized speedup measured in terms of FLOPS.

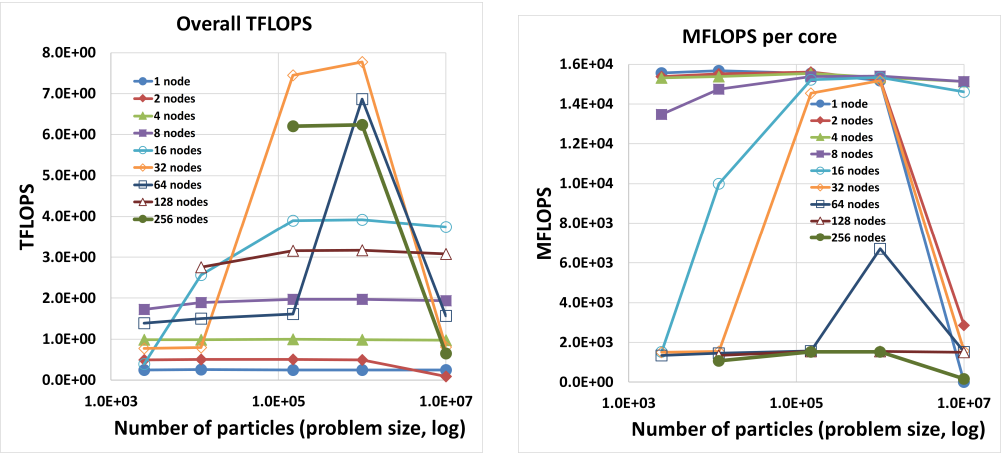
To use Eq. (6), it is suggested that the sequential speed is measured with a small problem size appropriate for one node processing, and parallel speedups are measured with large problem sizes appropriate for large number of nodes, thus a typical computational granularity (CG), i.e., 2.5k particles on a single node, is selected for the sequential speedup measurement. Table 11 presents the generalized speedup measurements across four orders of magnitude of simulation scale. Overall, the generalized speedup increases with increasing problem size.

Figure 16 plots the total and per core FLOPS performance with regard to problem size. The total FLOPS performance exhibits arched curves and indicates peak values at the problem size of 1.0E+5. Regarding the uniprocessor FLOPS performance, there is no clear trend. Yan and Regueiro (2018c) reveals the fact that different optimal computational granularity (CG), i.e.,

Table 11: Generalized speedup across various scales

Scale	2.5k	12k	150k	1M
TFLOPS	0.249	2.558	7.448	11.843
Speedup	1.0	10.3	29.9	47.5

number of particles per node, exists for different simulation scales to achieve minimum computational run time. When they are extracted at optimal or close-to-optimal computational granularity (CG), the FLOPS data reveal a clear trend, as well as the generalized speedup, as shown in Table 11.

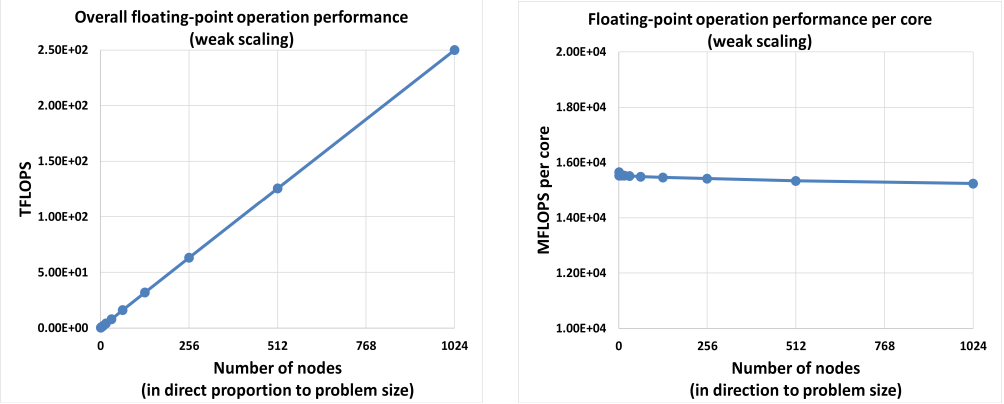


(a) Overall FLOPS. (b) Per core FLOPS.

Figure 16: Floating-point operation performance w.r.t problem size on Spirit (strong scaling).

Then we measure the FLOPS performance based on weak scaling tests, where the problem size (workload) assigned to each processor stays constant, namely, a constant computational granularity is employed across different orders of magnitude of simulation scale. In this test, 2,500 particles are assigned to each compute node (approximately 150 particles per core). 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 linear relationship with regard to the problem size, and the uniprocessor FLOPS performance nearly stays constant in spite of the problem size change.

Sandia experiments showed the uniprocessor performance is an increasing and asymptotic function of problem size for three practical and full-scale scientific problems: wave mechanics, fluid dynamics, and structural analysis (Gustafson, 1990; Gustafson et al., 1988). Unfortunately it is not observed in the 3D DEM problems of complex-shaped particles, whereby the uniprocessor performance is nearly a constant. Nevertheless, it does not hinder complex-shaped 3D DEM from delivering strong superlinear speedup. That means, both increasing uniprocessor performance or constant uniprocessor performance can contribute to superlinear speedup effect.



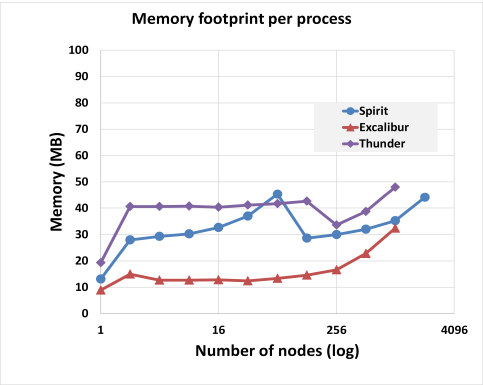
(a) Overall FLOPS. (b) Per core FLOPS.

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

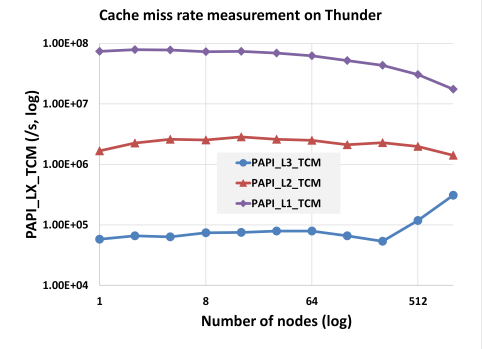
9. WEAK SCALING MEASUREMENT

Weak scaling measurements are also performed on the three DoD supercomputers. Figure 18(a) plots the memory usage per process at different simulation scales on the three supercomputers. Across all of the simulation scales it varies between 10∼50 MB, which is indeed a very low memory consumption, agreeing with the "high-CPU-low-memory" feature of 3D DEM for complex-shaped particles.

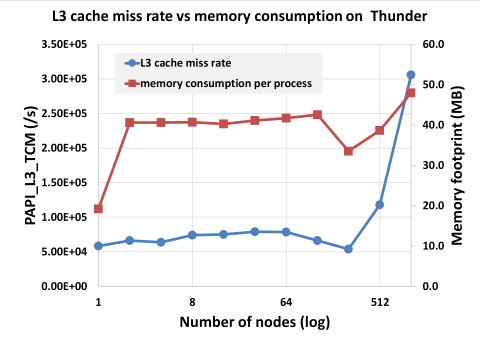
Figure 18(b) plots the L1, L2 and L3 cache miss rates of weak scaling measurement on Thunder. It is observed that `PAPI_L1_TCM` decreases and `PAPI_L2_TCM` nearly remains constant with respect to the workload that is in direct proportion to the number of compute nodes, and `PAPI_L3_TCM` increases at node number 512 and 1,024, in response to the memory consumption pattern shown in Figure 18(a). Figure 18(c) combines the `PAPI_L3_TCM`



(a) Memory footprint.



(b) Cache miss rates on Thunder.



(c) L3 cache miss rate on Thunder.

Figure 18: Weak scalability measurement

and memory footprint per process such that the matching pattern is demonstrated clearly. Again, it is seen that L3 cache has the most significant bearing among all of the three cache levels.

10. INFLUENCE OF NEIGHBOR SEARCH ALGORITHMS

As described in section 2.4, the $O(n^2)$ neighbor search algorithm executes faster than the $O(n)$ algorithm at fine CGs that are mostly employed in complex-shaped 3D DEM computational practice. Furthermore, Yan and Regueiro (2018a) studied the influence of the two algorithms on the superlinear speedup in complex-shaped 3D DEM: both $O(n^2)$ and $O(n)$ neighbor search algorithms exhibit a strong superlinear speedup on large scale simulations of complex-shaped 3D DEM, although the $O(n)$ algorithm

exhibits a lower speedup than the $O(n^2)$ algorithm across all computational scales and granularities.

11. CONCLUSION AND OUTLOOK

Strong superlinear speedup has been discovered in large scale simulations of parallel 3D DEM for complex-shaped particles, which features "high-CPU-low-memory" demand and CPU-bound rather than memory-bound, and the larger the scale, the stronger is the superlinear speedup. The phenomenon is reproduced on multiple DoD supercomputers. The strong scaling and weak scaling measurements show 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. The measurements are mostly attributed to the inherently perfect scalability of 3D DEM because its memory scalability function is a nonlinearly decreasing function of the number of processors.

It is worth pointing out that superlinear speedup exists in the low-memory-demand 3D DEM, aside from those high-memory-demand scientific computations that have been discovered. A constant uniprocessor FLOPS performance with regard to problem size can also contribute to the superlinear speedup, in addition to those cases with increasing uniprocessor FLOPS performance.

We emphasize that the superlinear speedup is commonplace for large scale complex-shaped 3D DEM, and DEM researchers should not hesitate to take advantage of this effect to speedup their simulations.

We expect to extend our analysis to and see similar superlinear speedup in other discrete methods widely used in engineering mechanics and material science, such as smoothed-particle hydrodynamics (SPH), Reproducing kernel particle methods (RKPM), PeriDynamics (PD), material point method (MPM), lattice Boltzmann methods (LBM), etc, across different computational length scales. As these methods do not require complex geometric contact resolution, their degree of superlinearity should vary depending on the computation ratio between neighbor search and core solver.

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.

Gene M Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485. ACM, 1967.

John W Baugh Jr and RKS Konduri. Discrete element modelling on a cluster of workstations. *Engineering with Computers*, 17(1):1–15, 2001.

Peter A Cundall and Otto DL Strack. A discrete numerical model for granular assemblies. *Geotechnique*, 29(1):47–65, 1979.

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.

Vance Faber, Olaf M Lubeck, and Andrew B White. Superlinear speedup of an efficient sequential algorithm is not possible. *Parallel Computing*, 3(3): 259–260, 1986.

Ian Foster. *Designing and building parallel programs*. Addison Wesley Publishing Company Reading, 1995.

Pengcheng Fu and Yannis F Dafalias. Fabric evolution within shear bands of granular materials and its relation to critical state theory. *International Journal for numerical and analytical methods in geomechanics*, 35(18): 1918–1948, 2011a.

Pengcheng Fu and Yannis F Dafalias. Study of anisotropic shear strength of granular materials using dem simulation. *International Journal for Numerical and Analytical Methods in Geomechanics*, 35(10):1098–1126, 2011b.

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.

John L Gustafson. Reevaluating amdahl's law. *Communications of the ACM*, 31(5):532–533, 1988.

John L Gustafson. Fixed time, tiered memory, and superlinear speedup. In *Proceedings of the Fifth Distributed Memory Computing Conference (DMCC5)*, pages 1255–1260, 1990.

John L Gustafson, Gary R Montry, and Robert E Benner. Development of parallel methods for a 1024-processor hypercube. *SIAM journal on Scientific and Statistical Computing*, 9(4):609–638, 1988.

David P Helmbold and Charles E McDowell. Modeling speedup (n) greater than n. *IEEE Transactions on Parallel & Distributed Systems*, (2):250–256, 1990.

David S Henty. Performance of hybrid message-passing and shared-memory parallelism for discrete element modeling. In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing*, page 10. IEEE Computer Society, 2000.

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.

Mark D Hill and Michael R Marty. Amdahl's law in the multicore era. *Computer*, (7):33–38, 2008.

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.

Alan H Karp and Horace P Flatt. Measuring parallel processor performance. *Communications of the ACM*, 33(5):539–543, 1990.

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.

Algirdas Maknickas, Arnas Kačeniauskas, Rimantas Kačianauskas, Robertas Balevičius, and Algis Džiugys. Parallel dem software for simulation of granular media. *Informatica*, 17(2):207–224, 2006.

J Quirm Michael. *Parallel Programming in C with MPI and OpenMP*. New York: McGraw-Hill Press, 2003.

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.

Dan I Moldovan. *Parallel processing from applications to systems*. Elsevier, 2014.

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.

Umpei Nagashima, Sachiko Hyugaji, Satoshi Sekiguchi, Mitsuhisa Sato, and Haruo Hosoya. An experience with super-linear speedup achieved by parallel computing on a workstation cluster: Parallel calculation of density of states of large scale cyclic polyacenes. *Parallel computing*, 21(9):1491–1504, 1995.

Tang-Tat Ng. Numerical simulations of granular soil using elliptical particles. *Comput. Geotech.*, 16(2):153 – 169, 1994. ISSN 0266-352X.

Tang-Tat Ng. Triaxial test simulations with discrete element method and hydrostatic boundaries. *Journal of Engineering Mechanics*, 130(10):1188 – 1194, 2004.

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.

Sasko Ristov and Marjan Gusev. Superlinear speedup for matrix multiplication. In *Information Technology Interfaces (ITI), Proceedings of the ITI 2012 34th International Conference on*, pages 499–504. IEEE, 2012.

Subhash Saini, Johnny Chang, and Haoqiang Jin. Performance evaluation of the intel sandy bridge based nasa pleiades using scientific and engineering applications. In *High Performance Computing Systems. Performance Modeling, Benchmarking and Simulation*, pages 25–51. Springer, 2013.

Yuan Shi. Reevaluating amdahls law and gustafsons law. *Computer Sciences Department, Temple University (MS: 38-24)*, 1996.

J Sienicki, P Agrawal, VD Agrawal, and ML Bushnell. Superlinear speedup in multiprocessing environment. In *Proceedings of the First International Workshop on Parallel Processing*, pages 261–265, 1994.

Xian-He Sun and Yong Chen. Reevaluating amdahls law in the multicore era. *Journal of Parallel and Distributed Computing*, 70(2):183–188, 2010.

Xian-He Sun and John L Gustafson. Toward a better parallel performance metric. *Parallel Computing*, 17(10-11):1093–1109, 1991.

Xian-He Sun and Lionel M Ni. Another view on parallel speedup. In *Supercomputing'90., Proceedings of*, pages 324–333. IEEE, 1990.

Xian-He Sun and Lionel M Ni. Scalable problems and memory-bounded speedup. *Journal of Parallel and Distributed Computing*, 19(1):27–37, 1993.

Vinodh Vedachalam and Davy Virdee. Discrete element modelling of granular snow particles using liggghts. *M. Sc., University of Edinburgh*, 2011.

David W Washington and Jay N Meegoda. Micro-mechanical simulation of geotechnical problems using massively parallel computers. *International journal for numerical and analytical methods in geomechanics*, 27 (14):1227–1234, 2003.

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.

Barry Wilkinson and Michael Allen. *Parallel programming*, volume 999. Prentice hall Upper Saddle River, NJ, 1999.

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 A. Regueiro. Comparison Between $O(n^2)$ and $O(n)$ Neighbor Search Algorithm and its Influence on Superlinear Speedup in Parallel 3D Discrete Element Method (DEM) for Complex-shaped Particles. In review, 2018a.

Beichuan Yan and Richard A. 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*, 2018b.

Beichuan Yan and Richard A. Regueiro. A Comprehensive Study of MPI Parallelism in Three-dimensional Discrete Element Method (DEM) for Complex-shaped Granular Materials. *Computational Particle Mechanics*, 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.