Parallelization of DEM simulation on distributed-memory computer via three-dimensional slice grid method

Kento YOKOO*, Masahiro KISHIDA* and Tsuyoshi YAMAMOTO*
* Department of Chemical Engineering, Kyushu University
744, Motoooka, Nishi-ku, Fukuoka-shi, Fukuoka, 819-0935, Japan
E-mail: yamamoto@chem-eng.kyushu-u.ac.jp

Received: 1 October 2020; Revised: 2 November 2020; Accepted: 10 November 2020

Abstract

A fluidized bed can efficiently filter dust particles, but its performance depends significantly on the fluidization state. To further develop the fluidized-bed filtration method, it is important to understand the filtration mechanisms in detail. Numerical simulation via the discrete element method is useful for solving these problems because the motion of each bed and dust particle is demonstrated. This system has large number of particles, and bias of the particle distribution is generated owing to the fluidization and supply of dust particles. Parallel computing on a distributed-memory computer is necessary to simulate many particles. Additionally, dynamic load balancing is a key technique for solving these problems. In this study, we developed a simple implementation of three-dimensional slice grid method and periodically used this method to balance the workload while keeping contact information such as the pair of colliding particles and its overlap. The computational efficiency of our method was assessed through an ideal problem involving a packed particle system and dust filtration in a fluidized bed. The changes in the particle number and particle distribution were examined. In the packed particle system, linear speed-up was obtained at particle number of 100 million and a message passing interface-process number of 1024. Moreover, the effectiveness of the dynamic domain decomposition method was confirmed by solving through the dust filtration problem.

Keywords: Discrete element method (DEM), Dynamic load balancing, Message passing interface (MPI) parallelization, Granular bed filter, Fluidized bed, Dust filtration

1. Introduction

The fluidized-bed filter was developed in our previous study and can filter PM$_{2.5}$ (i.e., particulate matter (PM) with a size of <2.5 µm) with a high collection efficiency (Yamamoto et al., 2016), although the collection efficiencies of the conventional devices are not sufficiently high for PM$_{2.5}$ filtration. Because the performance of this device depends significantly on the fluidization state, for practical use, it is important to understand the dust-particle filtration mechanisms in detail. However, it is difficult to observe dust particles in fluidized beds, because they are significantly smaller than the bed particles, and the number of bed particles is large. Thus, a computational fluid dynamics (CFD)-discrete element method (DEM) numerical simulation of this device should be performed. Because the number of particles in the system is large, distributed-memory computing with a message passing interface (MPI) is necessary to simulate it. The domain decomposition method is commonly used to give each processor a similar number of particles (Makino et al., 2004). Additionally, the dust particles are added from the bottom of this system, and the number of particles and the distribution of particles are affected by not only the fluidization but also the particle supply. Most of the dust particles are captured in the bottom bed particle layer, and the distribution of particles is further biased. Thus, domain decomposition should be performed dynamically. The slice grid method has the simplest implementation among the dynamic load-balancing methods. In previous studies, the two-dimensional slice grid method has mainly been used (Tsuzuki and Aoki, 2014; Furuichi et al., 2017; Nishiura et al., 2019). However, this system has a large spatial spread, and it is expected that the three-dimensional method is more suitable.

In this study, a simple implementation of a three-dimensional slice grid method that involves keeping tangential contact information, such as a pair of colliding particles and its overlap, was developed for DEM simulation. This
information must be maintained between MPI communication and sorting particles. The computational efficiency was assessed through an ideal problem involving a packed particle system and dust filtration in a fluidized bed. The changes in the particle number and particle distribution were examined. The major symbols used in this letter are presented in Table 1.

### Table 1 Symbols.

| Symbol | Description |
|--------|-------------|
| $A_p$ | Hamaker’s constant |
| $d_p$ | Particle diameter |
| $f_{k,i,j}$ | Contact force in $k$ direction between particles $i$ and $j$ |
| $F_{fg}$ | Gas–solid interaction term |
| $f_{vdw,i,j}$ | van der Waals force between particles $i$ and $j$ |
| $g$ | Gravitational acceleration |
| $h_{p,i,j}$ | Distance between surfaces of particles $i$ and $j$ |
| $I$ | Moment of inertia of particle |
| $m$ | Mass of particles |
| $N_x$, $N_y$, $N_z$ | Spatial division numbers in $x$, $y$, and $z$ directions, respectively |
| $N_{total}$ | Total number of particles |
| $p$ | Gas pressure |
| $T_{ij}$ | Torque between particles $i$ and $j$ |
| $u_g$ | Gas velocity |
| $v$ | Particle velocity |
| $v_k,i,j$ | Relative velocity in $k$ direction between particles $i$ and $j$ |
| $V_{cell}$ | Volume of CFD cell |
| $V_p$ | Particle volume |
| $\beta$ | Interphase momentum exchange coefficient |
| $\delta_{k,i,j}$ | Overlap in $k$ direction between particles $i$ and $j$ |
| $\Delta$ | Width of fine rectangular used in load balancing |
| $\varepsilon$ | Void fraction |
| $\tau$ | Stress tensor |
| $\omega$ | Angular velocity of particle |

2. Method

The numerical simulation method used in the present study is briefly explained in this section. The transitional and rotational motions of the particle are expressed using Newton’s second law in the DEM description.

\[
m_i \frac{dv_{ij}}{dt} = \sum_k (f_{n,i,j} + f_{t,i,j}) + m_i g \tag{1}
\]

\[
l_i \frac{d\omega_{ij}}{dt} = \sum_j T_{ij} \tag{2}
\]

\[
\delta_{t,i,j} = \int_{t_{start}}^{t} v_{t,i,j} dt \tag{3}
\]

Here, $i$ is the target particle, and $j$ is the neighbor particle; $v$ and $\omega$ are the velocity and angular velocity of the particle, respectively; $f_k,i,j = K_k \delta_{k,i,j} + \eta_k v_{k,i,j}$ is the $k$ direction contact force (where $k$ is $n$ for the normal direction, and $k$ is $t$ for the tangential direction), and it is modeled by a spring-dashpot system; $K_k$ and $\eta_k$ are the spring coefficient and dashpot coefficient; $\delta_{k,i,j}$ and $v_{k,i,j}$ are the $k$ direction relative velocity and overlap. Tangential overlap $\delta_{t,i,j}$ is the time integration of $v_{t,i,j}$ as Eq. (3), and $t_{start}$ represents the beginning time when particle $i$ comes into contact with particle $j$. 

[DOI: 10.1299/mel.20-00444] 
© 2020 The Japan Society of Mechanical Engineers
The spring and dashpot coefficients are determined by the Young’s modulus, shear modulus, Poisson’s ratio, coefficient of restitution, density, and particle diameter. $T_{ij}$ represents the torque between $i$ and $j$. Our simulation code was parallelized via the flat MPI. The calculations were performed at the Research Institute for Information Technology of Kyushu University.

3. Implementation
3.1 Spatial decomposition

The unbalanced workload is generated via a static domain decomposition method for the system with moving particles or supplying particles because of the bias of the particle distribution due to the particle motion and the changes in the number of particles. The numerical domain is re-decomposed to overcome the unbalanced load during the DEM simulation. In our domain decomposition method, the numerical domain was divided into $N_e$ pieces in $x$ coordinate to assign each piece almost the same number of particles, and each piece was further divided into $N_c$ pieces in $y$ and $z$ coordinates. Here, $N_e$, $N_y$, and $N_z$ were arbitrarily set under the condition of $N_eN_yN_z = $MPI-process number. The processors identified as a 3-tuple $(l,m,n)$ handled the sub-domain of $l$th in the $x$ direction, the $m$th in the $y$ direction, and the $n$th in the $z$ direction $(0 \leq l \leq N_e-1, 0 \leq m \leq N_y-1, 0 \leq n \leq N_z-1)$. Additionally, $(l,0,0)$–$(l,N_y-1,N_z-1)$ represents a set of processors handling the sub-domains of $l$th in the $x$ direction whose $m$ and $n$ of a 3-tuple $(l,m,n)$ can be any value. $(l,m,0)$–$(l,N_y-1,N_z)$ represents a set of processors handling the sub-domains of $l$th in the $x$ direction and the $m$th in the $y$ direction whose $n$ of a 3-tuple $(l,m,n)$ can be any value. The protocol is described in detail below.

Figure 1 shows a schematic of the domain decomposition in the $x$ direction. For division into $N_c$ pieces, the whole domain is sliced into rectangles of width $\Delta$, which is the cutoff length for interaction in the system. The number of particles included in each rectangle is counted in each processor, and this information was gathered for all the processors. The $l$th sub-domain is determined to be the $l$th group of continuous rectangles in $x$ coordinate that contains about $N_{total}/N_e$ particles. The processors of $(l,0,0)$–$(l,N_y-1,N_z-1)$, which handle the $l$th sub-domain, are registered as a new MPI communicator Group $x$. Because the width of the $l$th sub-domain is changed by the domain re-decomposition in the $x$ direction, some particles, which were assigned to processor $(l,m,n)$ before the re-decomposition, may belong to the left $(l-1)$th or right $(l+1)$th neighboring sub-domain after the re-decomposition. To balance the workload in the $x$ coordinate, processor $(l,m,n)$ sends the particles located outside the $l$th sub-domain to processor $(l+1,m,n)$ or $(l-1,m,n)$ via MPI communication. The same operation is conducted for each sub-domain in the $y$ and $z$ directions.

![Fig. 1 Schematic of domain decomposition for load balancing in the x direction.](image)

To divide the sub-domain into $N_y$ pieces in the $y$ direction, the sub-domain is sliced into rectangles of depth $\Delta$. The number of particles in each rectangle is counted, and the number of particles is gathered in Group $x$. The $m$th sub-domain
is the $m$th group of continuous rectangles in the $y$ coordinate, including about $N_{total}/N_x N_y$ particles. Processors of $(l,m,0)$–$(l,m,N_y-1)$, which treat this sub-domain, are registered as a new communicator Group $y$. To balance the workload in the $y$ direction, the MPI communication should be performed as in the $x$ direction. Processor $(l,m,n)$ sends a particle located outside the $m$th sub-domain to processor $(l,m+1,n)$ or $(l,m-1,n)$. The sub-domain is further decomposed in the $z$ direction. It is sliced into rectangles of height $\Delta$. The number of particles in each rectangle is determined, and this information is gathered in Group $y$. The $n$th sub-domain is the $n$th group of continuous rectangles in $z$ coordinates that contains about $N_{total}/N_x N_y N_z$ particles. Processor $(l,m,n)$ handles this sub-domain and sends particles located outside the $n$th sub-domain to $(l,m,n+1)$ or $(l,m,n-1)$ for load balancing, similar to the $x$ and $y$ directions.

### 3.2 Keeping tangential contact information

The contact information of the previous timestep, such as the pair of colliding particles and their tangential overlap, should be retained before and after the MPI communication and particle sorting to calculate the tangential overlap of Eq. (3) and the contact force. As a precondition, we assign particle $i$ the serial number Particle ID[$i$] across all the processors. The number of particles $j$ contacting particle $i$ ($i<j$) is determined, and this number is stored in an array called Contact number. The particle ID of particle $j$ colliding with particle $i$ ($i<j$) and its tangential overlap are stored in arrays called List of contact ID and Tangent overlap. Here, the contact information does not need to be stored for both particle $i$ and particle $j$, because we can know the complete contact information of either particle $i$ or $j$. Figure 2 shows a schematic of the contact information after domain re-decomposition. The detailed protocol is described by Algorithm 1. Because Tangent overlap is treated as the List of contact ID, it is abbreviated. The contact information is temporarily stored in Contact number tmp and List of contact ID tmp. This information is sent to another processor, and the received information is temporally stored in Contact number_rcv and List of contact ID_rcv via the MPI_SENDRECV function. The contact information, which is originally possessed by or received from another processor, is re-stored into Contact number and List of contact ID by eliminating information sent to another processor.

![Fig. 2 Schematic of storing contact information and its communication to keep this information before and after domain re-decomposition. Each particle has an ID across all processors, and the number and ID of a colliding pair are stored in Contact number and List of contact ID. The communication of contact information is shown in red.](image)

### 3.3 Communication for interactions

Some particles assigned to different sub-domains interact in the domain decomposition method. The information of particles that are near the boundary of a sub-domain should be sent to adjacent sub-domains sharing the same boundary. Although the adjacent sub-domains are fixed in static domain decomposition, our dynamic load balancing changes the adjacent sub-domains in the $x$ and $y$ coordinates with variations in the particle distribution. In contrast, there are no discrepancies in the $z$ direction in the decomposition method. Figure 3 shows the geometric relationship of decomposed sub-domains and a schematic of the communication in the $x$, $y$, and $z$ directions. The arrows indicate the sending and receiving of particle information. To detect sub-domains sharing the boundary with sub-domain $(l,m,n)$, processor $(l,m,n)$ gathers information about the positions of the right ($(l+1)^{th}$ in the $x$ direction), left ($(l-1)^{th}$), backward ($(m+1)^{th}$ in the $y$ direction), and forward ($(m-1)^{th}$) sub-domains. In contrast, the top and bottom neighboring sub-domains $(l,m,n+1)$ and $(l,m,n-1)$ are fixed because there are no discrepancies of the sub-domains in the $z$ direction. Processor $(l,m,n)$
communicates with the detected sub-domains as follows.

Processor \((l,m,n)\) sends information of particles near the boundary (i.e., within the cutoff length of the interaction from the boundary) to the left neighboring sub-domains as real arrows, as shown in Fig. 3. Thus, processor \((l,m,n)\) also receives information from the right neighboring sub-domains. Remarkably, processor \((l,m,n)\) need not send information to the right side. The forces between the particles assigned to \((l,m,n)\) and the particles received from the right are calculated by \((l,m,n)\). The calculated forces acting on received particle are sent to the right via the pathway opposite to the arrow shown in Fig. 3. Complete information on the interactions is obtained through communication. In the \(y\) direction, processor \((l,m,n)\) sends forward information of particles that are originally included in \((l,m,n)\) or received from the right. In contrast to the case of the \(x\) direction, processor \((l,m,n)\) also sends information of particles that are received from the right neighbor to backward neighbor, as indicated by the dotted arrow in Fig. 3. In the \(z\) direction, processor \((l,m,n)\) sends information of particles that are originally included in \((l,m,n)\) or received from all adjacent sub-domains in the \(x\) and \(y\) directions to the bottom neighbor. It also sends information of particles received from the right or backward to the top neighbor. The interactions are calculated using the information gathered by MPI communications in each processor. Furthermore, we use the grid search method (Watanabe et al., 2011) for efficient computation. In this method, sub-domain \((l,m,n)\) is divided into cubes, whose size is the cutoff length of interaction, for reducing the computational complexity of \(O(N^2)\) to \(O(N)\), where \(N\) represents the number of particles in the sub-domain. The particles were registered to each cube. The interaction pair to the target particle is searched only in the cubes adjacent to the cube contains target particle.

![Fig. 3 Schematic of MPI communication for the particle information in the \(x\), \(y\), and \(z\) coordinates. Each processor gathers information about the position of neighboring sub-domains and communicates with sub-domains sharing the same boundary. The communication of particle information is indicated by arrows.](image)

### 4. Results and discussion

To evaluate the computational efficiency, we conducted two types of simulations of packed particles and dust filtration in a fluidized bed. As is generally known, the parallel-computing efficiency of a small system becomes saturated at a small process number. We handled three cases of packed particles: (1) the number of particles was 0.05 million, and the size of the numerical domain was \(2\times8\times50\ \text{mm}\); (2) the number of particles was 10 million, and the size of the numerical domain was \(12\times47\times300\ \text{mm}\); (3) the number of particles was 100 million, and the size of the numerical domain was \(25\times100\times630\ \text{mm}\). The size of the numerical domain increased according to the number of particles, almost maintaining the aspect ratio. The particle diameter was 420 \(\mu\text{m}\), the density was 3900 \(\text{kg/m}^3\), the Young’s modulus was 1 GPa, the Poisson’s ratio was 0.25, and the coefficient of restitution was 0.8. Figure 4 shows the variation of the computing coefficient with respect to the number of processes. The ratio of the number of particles to the maximum time for one timestep was used to assess the parallel-computing efficiency (Nishiura et al., 2017), as shown in Eq. (4). For 0.05 million particles, linear speed-up was observed at only <16 MPI processes, and it was approximately saturated at >124 processes. For 10 million particles, linear speed-up was observed at <124 MPI processes. For 100 million particles, linear speed-up was observed at <1024 MPI processes.
Performance = \frac{\text{Particle number in whole domain}}{\text{Elapsed time for one time-step of particle simulation}}

Algorithm 1 MPI communication retaining contact information

for all \( i \) particle
\[ j++; \]
\[
\begin{align*}
\text{Contact Number}_{\text{tmp}}[j] &= \text{Contact Number}[i]; \\
\text{for} \ (k; \ k < \text{Contact Number}[i]; k++) \\
& m++; \ n++; \\
\text{List of contact ID}_{\text{tmp}}[n] &= \text{List of contact ID}[m] \\
\text{elseif} \ (\text{particle } i \text{ is not sent}) \ m = m + \text{Contact Number}[i]
\end{align*}
\]

MPI\_SENDRECV(\text{Contact number}_{\text{tmp}}, \ldots, \text{Contact number}_{\text{recv}}, \ldots)

MPI\_SENDRECV(\text{List of contact ID}_{\text{tmp}}, \ldots, \text{List of contact ID}_{\text{recv}}, \ldots)

for all \( i \) particle and received particles
\[ j++; \]
\[
\begin{align*}
\text{Contact number}[j] &= \text{Contact Number}[i] \\
\text{for} \ (k; \ k < \text{Contact number}[i]; k++) \\
& m++; \ n++ \\
\text{List of contact ID}[n] &= \text{List of contact ID}[m] \\
\text{elseif} \ (\text{particle } i \text{ is not sent}) \ n = n + \text{Contact number}[i]
\end{align*}
\]

Fig. 4 MPI parallel-computing efficiency. The performance is plotted with squares for 100 million particles, circles for 10 million particles, and triangles for 0.05 million particles. At 100 million particles, the computational performance increased linearly with the increasing number of central processing units (CPUs) until the number of MPI processes reached 1024.

For dust filtration, the DEM simulation code was coupled with in-house CFD code. The incompressible gas dynamics in a fluidized bed are described by the locally averaged Navier–Stokes equations (Tsuji et al., 1993), and these equations were solved using the semi-implicit method for pressure-linked equation (SIMPLE) method.

\[
\frac{\partial \varepsilon_i}{\partial t} + \nabla \cdot (\varepsilon_i \rho_j \mathbf{u}_j) = 0
\]
\[
\frac{\partial (\varepsilon p u_f)}{\partial t} + \nabla \cdot (\varepsilon p u_f; u_f) = -\nabla p - F_{pf} + \nabla \cdot (\varepsilon \tau) + \varepsilon p g
\]  
(6)

\[
F_{pf} = \frac{\sum_{\text{particle in cell}} \beta (v_p - u_f)}{V_{cell}}
\]  
(7)

\[
f_{pf} = \frac{v_p \beta}{(1-\varepsilon)} (u_f - v_p)
\]  
(8)

Here, \( \rho_f \) and \( \varepsilon \) represent the gas density and void fraction, respectively; \( u_f \) and \( p \) represent the gas velocity and pressure of the gas phase, respectively; \( \tau \) is the stress tensor; and \( g \) represents the gravitational acceleration. \( F_{pf} \) is the gas–solid interaction term for the gas phase. \( V_{cell} \) represents the volume of the CFD cell. \( \beta \) is an interphase momentum exchange coefficient, which is calculated using the Ergun and Wen–Yu drag correlations. \( f_{pf} \) is the gas–solid interaction term for the particles. \( V_p \) represents the particle volume. The gas species is air, and the superficial velocity was set as 0.4 m/s. The CFD grid size was \( 2 \times 2 \times 2 \) mm\(^3\). The fluidized bed was constructed with rectangles having dimensions of \( 2 \times 8 \times 50 \) mm\(^3\) and 0.05 million particles. The particle properties were identical to those used in the computation of the packed particle system. Hamaker’s constant for the particles was \( 16.5 \times 10^{-20} \) J. The dust particle collection is occurred via only the adhesion force between dust particles and bed particles (i.e., van der Waals force). The van der Waals force \( f_{vdw,ij} \) was calculated using Eq. (9) (Kobayashi et al., 2013), and Hamaker’s constant determined the van der Waals force. The dust particle size was 2 \( \mu \)m, the density was 1900 kg/m\(^3\), the Young’s modulus was 0.01 GPa, the Poisson’s ratio was 0.2, and the Hamaker’s constant was \( 5 \times 10^{-20} \) J.

\[
f_{vdw,ij} = \frac{A^*_{ij} d^*_p}{12 h_{ij}}
\]  
(9)

\( A^*_{ij} = (A_p A_d)^{1/2} \) and \( d^*_p = d_p (d_p + d_d) \) are the effective values of the Hamaker’s constants and the diameters of particles \( i \) and \( j \), respectively. \( A_p \) and \( d_p \) represent the Hamaker constant and the particle diameter, respectively. \( h_{ij} \) represents the distance between the surfaces of particles \( i \) and \( j \). Dust particles were added from bottom with gas at rate of 1000/0.01 s. The total simulation time was 0.5 s. The number of MPI processes was 128.

Figure 5(a) shows domain decomposition at 0 s. Figures 5(b) and (c) show the domain decomposition during dust filtration in a fluidized bed at 0.5 s with dynamic and static load balancing. The color of the particles is determined by the process number. The static method does not re-divide the numerical domain, and the other operation is identical to that in the dynamic method. The dust was filtered at the bottom layer of the fluidized bed via the adhesion force. The collection efficiency was 98.5%, and the high collection efficiency was reproduced from this simulation (Yamamoto et al., 2016). The dust particle distribution at 0.5 s is shown in Fig. 5(d). It biased the distribution of particles downward in this system. The distribution of particles was also biased, owing to fluidization. Thus, the sub-domain of the bottom side was very thin under dynamic load balancing, for giving each processor almost same load. The difference of number of particles among these processors is less than 10% of average number of particles in each processor. On the other hand, the difference is more than ten times of average particle number in each process with static load-balancing. The performances of particle simulation with static and dynamic load balancing, which is evaluated via Eq. (4), are \( 2.03 \times 10^7 \) and \( 0.98 \times 10^7 \). Moreover, the performance of the dynamic load-balancing is as almost the same as the performance without dust particle of \( 2.02 \times 10^7 \) and is maintained despite of dust particle supply. It indicates effectiveness of our domain decomposition for the variation of particle number and particle distribution.

5. Conclusions

A simple implementation of flat-MPI parallelization for DEM simulation is proposed. The numerical domain was periodically re-decomposed in three dimensions during computation, and particle information was communicated while maintaining tangential contact information to obtain a balanced workload. Linear speed-up was obtained at a particle number of 100 million and 1024 MPI processes through the packed-particle problem. The effectiveness of our method for variation in particle number and distribution was shown through the dust filtration problem. The computational efficiency was maintained with and without dust-particle filtration with our dynamic load-balancing method, and highly
efficient PM collection was reproduced.

Fig. 5 Domain decomposition with dynamic and static load balancing: (a) initial state; (b) dynamic load balancing; (c) static load balancing; (d) distribution of dust particles. Dust particles are filtered in the bottom layer by the adhesion force, which biases the particle distribution in the system. In the dynamic load balancing, the sub-domain of the bottom side (red) becomes very thin compared with the initial state.

Acknowledgements

This study was partially supported by the Environmental Research and Technology Development Fund JPMEERF20191007 of the Environmental Restoration and Conservation Agency of Japan.

References

Furuichi, M. and Nishiura, D., Iterative load-balancing method with multigrid level relaxation for particle simulation with short-range interactions, Computer Physics Communications, Vol. 219 (2017), pp 135-148, DOI:10.1016/j.cpc.2017.05.015

Junichiro, M., A Fast Parallel Treecode with GRAPE, Publications of the Astronomical Society of Japan, Vol. 56, No. 3 (2004), pp 521-531, DOI:10.1093/pasj/56.3.521

Kobayashi, T., Tanaka, T., Shimada, N., Kawaguchi, T., DEM–CFD analysis of fluidization behavior of Geldart Group A particles using a dynamic adhesion force model, Powder Technology, Vol. 248 (2013), pp. 143-152, DOI:10.1016/j.powtec.2013.02.028

Nishiura, D., Furuichi, M., Sakaguchi, H., Real-Scale DEM Simulations on the Fault Evolution Process Observed in Sandbox Experiments, Journal of the Society of Powder Technology, Vol. 56, No. 56 (2019), pp. 203-210, DOI:10.4164/spj.56.203

Tsuzuki, S. and Aoki, T., Large-scale granular simulations using Dynamic load balance on a GPU supercomputer, Proceedings of the 2014 ACM/IEEE conference on Supercomputing (SC’14), (2014)

Yamamoto, T., Hori, K., Tatebayashi, J., An experimental investigation of the PM adhesion characteristics in a fluidized bed type PM removal device, Powder Technology, Vol. 289, pp 31-36, DOI:10.1016/j.powtec.2015.11.046

Watanabe, H., Suzuki, M., Ito, N., Efficient Implementations of Molecular Dynamics Simulations for Lennard-Jones Systems, Progress of Theoretical Physics, Vol. 126, No. 2 (2011), pp. 203-235, DOI: 10.1143/PTP.126.203