Springer

KSME

Journal of Mechanical Science and Technology 27 (4) (2013) 973~980
www.springerlink.com/content/1738-494x
DOI 10.1007/s12206-013-0226-4

GPU-based collision analysis between a multi-body system and numerous particles †

Hye-Young Jung[1], Chul-Woong Jun[1] and Jeong-Hyun Sohn[2,*]

[1]*Graduate School of Mechatronics Engineering, Pukyong National University, Busan, 608-739, Korea*
[2]*Department of Mechanical and Automotive Engineering, Pukyong National University, Busan, 608-739, Korea*

(Manuscript Received August 18, 2012; Revised October 10, 2012; Accepted November 19, 2012)

Abstract

The use of a graphics processing unit (GPU) is an ideal solution for problems on data-parallel computations. The serial CPU-based program for collision analysis between a multi-body system and numerous particles is rebuilt as a parallel program that uses the advantages of a GPU. In this study, a GPU is used to effectively perform multi-body dynamic simulation with particle dynamics. The multi-body system has 20 circular objects, 19 spring-damper force elements, and 2 revolute joints. The motion equations are formulated using the Cartesian coordinate system, and the implicit Hilber-Hughes-Taylor integration algorithm is used for the integral equation. To detect collisions between a multi-body system and particles or between particles, a spatial subdivision algorithm and a discrete element modeling are used. The developed program is verified by comparing the results with ADAMS. The numerical efficiencies of the serial program using CPU and the parallel program using GPU are compared according to the number of particles. The results show that the greater the number of particles, the more computing time can be saved. For example, when the number of particles is 900, the computing speed of the parallel analysis program is about five times faster than that of the serial analysis program.

Keywords: Graphics processor unit; Parallel programming; Multi-body dynamics; Spatial subdivision; Discrete element method

1. Introduction

A graphics processing unit (GPU) is an arithmetic unit that specializes in computer graphics. GPU makes the CPU process computer data faster by dealing with the three-dimensional (3D) computer graphics by itself. As each graphics card comprises a board, memory, and GPU, the performance and size of the graphics card can be freely upgraded. However, in the CPU, the main board and memory are produced by other manufacturers. Thus, changing CPU performance is not easy. To address this problem, GPUs have been developed by increasing the quantity of their cores to make them capable of processing massive computer data, which in turn allows them to generate more gorgeous and natural 3D images. The programmable GPU has evolved in recent years, becoming highly parallel and multithreaded. Numerous core processors with tremendous computational power and very high memory bandwidth have been developed to meet the increasing market demand for real time, high-definition 3D graphics. Although existing GPUs have been used for processing simple graphics, a general-purpose GPU (GPGPU) is being developed for public operation, which is not the original purpose of GPUs. As the API function handles the graphics engine, programmers and beginners found the GPGPU difficult to use, and flexible programs could not be developed because of the limited DRAM memory bandwidth that generates bottleneck conditions. To solve many complex computational problems more efficiently without using a CPU, NVIDIA, on November 2006, introduced CUDA™, which is a general-purpose parallel computing architecture with a new parallel programming model. CUDA also has an instruction set architecture that powers the parallel computing engine in the NVIDIA GPU. If the strengths of GPUs, including their many processors, wide bandwidth, and quick operation, are applied to real-time operations using parallel programming, obtaining efficient results is possible. Currently, NVIDIA is producing CUDA architecture-based graphics hardware to supply products that are capable of providing both fast computation and quality graphics to consumers [1-3]. To develop a general-purpose program that uses a GPU, NVIDIA is supplying not only graphics hardware but also an integration environment that includes "program model," "language," "compiler," "library," "debugger," and "profiler." Using CUDA programming, various studies are being conducted in several areas such as in entertainment, industrial, design, medicine, education, and finance. Many successful computing examples are thus being put forward [4].

Iterations are unavoidable in performing serial CPU-based

*Corresponding author. Tel.: +82 51 629 6166, Fax.: +82 51 629 6150
E-mail address: jhsohn@pknu.ac.kr
† Recommended by Editor Yeon June Kang

analysis for multi-body dynamics. Iterations make real-time analysis difficult as the system size increases. As the analysis of the dynamic behavior of a gigantic multi-body system requires fast computing, we can replace iterative computing with parallel computing. Doing so speeds up computing time. Thus, studies using GPU parallel computing for multi-body dynamics are increasing. Dan et al. proposed an approach to GPU dynamic simulation wherein large collections of rigid bodies mutually interact through millions of frictional contacts [5, 6]. However, the GPU parallelization method and the effects of increasing multi-bodies and force elements on parallel computing are not clearly understood.

This present study proposes a systematic approach to parallel computing algorithm that uses GPU for the collision analysis of a multi-body system and numerous particles. To calculate the contact force between the modeled multi-body system and a particle, a spatial subdivision algorithm [7, 8] and a discrete element modeling (DEM) [9] are used. Motion equations for multi-body dynamics are derived using Cartesian coordinates [10, 11]. The implicit integration algorithm called the Hilber-Hughes-Taylor (HHT) algorithm is used for the analysis. The data interface between the CPU and GPU is also explained. The computing time of the serial CPU-based program is compared with that of the parallel CUDA-based program to study the numerical efficiency of the GPU.

2. Dynamic analysis method for multi-body system

The HHT or HHT-α method [12-14] is widely used in structural dynamics for the numerical integration of a linear set of second order differential equations. A precursor of the HHT method is the newmark method, wherein a family of integration formulas that depend on two parameters (β and γ) is presented as Eqs. (1) and (2).

$$q_{n+1} = q_n + h\dot{q}_n + \frac{h^2}{2}\left[(1-2\beta)\ddot{q}_n + 2\beta\ddot{q}_{n+1}\right] \tag{1}$$

$$\dot{q}_{n+1} = \dot{q}_n + h\left[(1-\gamma)\ddot{q}_n + \gamma\ddot{q}_{n+1}\right]. \tag{2}$$

The HHT method possesses the stability and order properties provided by $\alpha \in [-1/3,\ 0]$ and Eq. (3).

$$\gamma = \frac{1-2\alpha}{2} \qquad \beta = \frac{(1-\alpha)^2}{4}. \tag{3}$$

In any constrained mechanical system, the joints that connect bodies restrict their relative motion and impose constraints on the generalized coordinates. Kinematic constraints are then formulated as algebraic expressions (4) involving generalized coordinates [10, 11], where m is the total number of independent constraint equations that must be satisfied by the generalized coordinates throughout the simulation. Differentiating Eq. (4) with respect to time results in the velocity kinematic constraint equation, as shown in Eq. (5). The overdot denotes the differentiation with respect to time, and the subscript denotes partial differentiation, wherein $1 \le i \le m$, $1 \le j \le p$. The acceleration kinematic constraint equation in Eq. (6) is obtained by differentiating Eq. (5) with respect to time. System time evolution is controlled by the Lagrange multiplier form of the constrained equations of motion (7), where $M(q) \in R^{p\times p}$ is the generalized mass, and $Q(\dot{q},\ q,\ t) \in R^p$ is the action (as opposed to the reaction $\Phi_q^T(q)\lambda$) force acting on the generalized coordinates. $q \in R^p$ and $\lambda \in R^m$ are the Lagrange multipliers associated with the kinematic constraints.

$$\Phi(q,t) = \left[\Phi_1(q,t) \quad \cdots \quad \Phi_m(q,t)\right]^T = 0 \tag{4}$$

$$\Phi_q(q,t)\dot{q} + \Phi_t(q,t) = 0 \qquad \Phi_q = \left[\frac{\partial\Phi_i}{\partial\Phi_j}\right] \tag{5}$$

$$\Phi_q(q,t)\ddot{q} + \left(\Phi_q(q,t)\dot{q}\right)_q\dot{q} + 2\Phi_{qt}(q,t)\dot{q} + \Phi_{tt}(q,t) = 0 \tag{6}$$

$$M(q)\ddot{q} + \Phi_q^T(q)\lambda = Q(\dot{q},q,t). \tag{7}$$

Note that Eq. (7) can also be written as Eq. (8), where H represents the generalized forces that include the constraint forces. Eq. (8) at time step t_{n+1} can be written as Eq. (9). The right side of Eq. (9) can be written as Eq. (10) using a Taylor series expansion. t_m is the time between t_n and τ_{n+1}, and $\dot{H}(t_m)$ can be approximated by assuming linear change in H, that is, Eq. (11). By substituting this equation into Eq. (10) and by using the resulting expression $H(\tau_{n+1})$ in Eq. (9), Eq. (12) is obtained. Using Eq. (8), Eq. (12) can then be written as Eq. (13).

$$M\ddot{q} = Q - C_q^T\lambda = H \tag{8}$$

$$\left(M\ddot{q}\right)_{n+1} = H\left(\tau_{n+1}\right) \tag{9}$$

$$H\left(\tau_{n+1}\right) = H\left(t_n\right) + (1+\alpha)h\dot{H}(t_m) \tag{10}$$

$$\dot{H}\left(t_m\right) = \left(H\left(t_{n+1}\right) - H\left(t_n\right)\right)/h \tag{11}$$

$$\left(M\ddot{q}\right)_{n+1} = (1+\alpha)H\left(t_{n+1}\right) - \alpha H\left(t_n\right) \tag{12}$$

$$\left(M\ddot{q}\right)_{n+1} + (1+\alpha)\left(C_q^T\lambda - Q\right)_{n+1} - \alpha\left(C_q^T\lambda - Q\right)_n = 0. \tag{13}$$

To obtain $\ddot{q}_{n+1}$ and λ_{n+1}, the iterative Newton-Raphson method should be used. This method requires constructing and solving the system of Eq. (14) at iteration k, where Δ indicates Newton differences, and matrix M, e_1, and e_2 are defined as Eqs. (15)-(17), respectively.

$$\begin{bmatrix}\hat{M} & \Phi_q^T\\ \Phi_q & 0\end{bmatrix}\begin{bmatrix}\Delta\ddot{q}\\ \Delta\lambda\end{bmatrix}^{(k)} = \begin{bmatrix}-e_1\\ -e_2\end{bmatrix}^{(k)} \tag{14}$$

$$\hat{M} = \frac{\partial e_1}{\partial\ddot{q}} = \frac{1}{1+\alpha}\left(M\ddot{q}\right)_{n+1} - h\gamma\frac{\partial Q}{\partial\dot{q}} + \left[\frac{1}{1+\alpha}\left(M\ddot{q}\right)_q + \left(\Phi_q^T\lambda\right)_q - \frac{\partial Q}{\partial q}\right]\beta h^2 \tag{15}$$

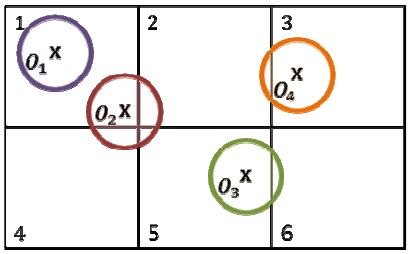

Fig. 1. Example of spatial subdivision of four objects.

$$e_1 = \frac{1}{1+\alpha}\left(M\ddot{q}\right)_{n+1} + \left(\Phi_q^T\lambda - Q\right)_{n+1} - \frac{\alpha}{1+\alpha}\left(\Phi_q^T\lambda - Q\right)_n \quad (16)$$

$$e_2 = \frac{1}{\beta h^2}\Phi(q,t)\,. \quad (17)$$

3. Contact modeling for collision analysis

3.1 Spatial subdivision

Most efficient collision detection implementations use a two-phase approach: a broad phase followed by a narrow phase. In the broad phase, collision tests are usually based on bounding volumes only, but such tests are conducted quickly to immediately prune away object pairs that do not collide with each other [7]. The broad phase output is the potentially colliding set of object pairs. In the narrow phase, collision tests are performed only for pairs of the potentially colliding set [8]. Such an approach is very suitable when numerous objects are actually not colliding. Many object pairs are thus rejected during the broad phase. In this study, spatial subdivision, which is one of the broad-phase algorithms, is used. Spatial subdivision partitions space into a uniform grid such that a grid cell is at least as large as the largest object. Each cell contains a list of each object whose centroid is within that cell. A collision test between two objects is only performed if they appear in the same cell and if at least one of them has its centroid in the given cell. For example, as shown in Fig. 1, a collision test is performed between objects O_1 and O_2 because both have their centroids in cell 1, and between objects O_2 and O_3 because both appear in cell 5, and O_3 has its centroid in cell 5. However, no collision test is performed between O_2 and O_4 because both appear in cell 2, but neither of their centroids is in cell 2. In this study, the simulation space is a square with a 6 m x 6 m dimension. Each edge is divided into 60 equal parts because all particles are designed to have a 0.05 m radius. The cell size is 0.1 m, and the square has 3600 (60×60) cells.

3.2 Discrete element method

To calculate the contact force between circular bodies, the DEM is used [9]. The DEM is one of the numerical methods used to calculate the motion of mass particles like sand. The DEM is also known as the best method to simulate particle behavior. This method is originally devised for the rock mechanics problem, but it has been applied to all granular material problems since 1979. Nowadays, the DEM is used in many industries. Fig. 2 shows the flow chart for the particle dynamics simulation using the DEM.

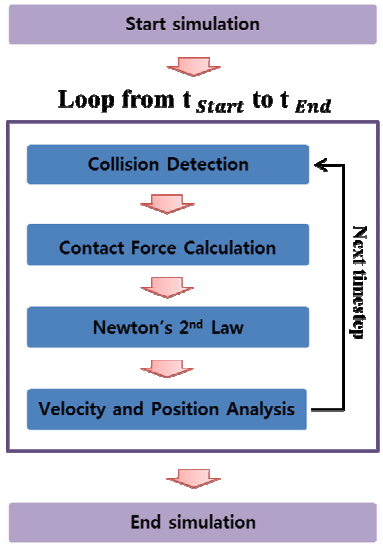

Fig. 2. Flow chart for particle dynamics simulation using the DEM.

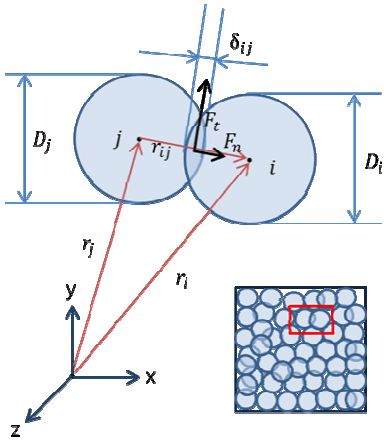

Fig. 3. Contact force calculation between two particles.

During particle dynamics simulation, four stages are repeated until simulation time ends. The first stage involves detecting pairs of colliding particles, and the next stage involves calculating the contact force based on the Kelvin-Voigt contact model [15]. The contact model is composed of spring and damping forces.

Fig. 3 shows the contact force calculation between two particles. Eq. (18) represents the contact force, where parameter K is the spring coefficient, and C is the damping coefficient. In Eq. (19), r_{ij} (the relative position between the i-th particle and the j-th particle) can be calculated by subtracting the j-th particle position from the i-th particle position. From this relative position vector, the distance between two particles is

computed using Eq. (20). The normal direction unit vector n_{ij} should be calculated using Eq. (21). Eq. (22) represents the relative velocity of the normal direction.

$$F = K\delta^n + C\dot{\delta} \tag{18}$$

$$r_{ij} = r_i - r_j \tag{19}$$

$$d = \sqrt{r_{ij} \cdot r_{ij}} \tag{20}$$

$$n_{ij} = \frac{r_{ij}}{d} \tag{21}$$

$$\dot{\delta}_{ij} = v_i - v_j \tag{22}$$

$$d_c = \frac{D_i}{2} + \frac{D_j}{2} \tag{23}$$

$$\delta_{ij} = d_c - d \tag{24}$$

where d_c means is the sum of the i-th particle radius and the j-th particle radius in Eq. (23); δ_{ij} is the penetration depth of the i-particle and j-particle in Eq. (24). If the distance d is shorter than d_c, the penetration depth and contact force are computed.

Next, Newton's second law is used to calculate particle accelerations, and the HHT method is used to compute the velocity and position of all particles [14].

4. Parallel programming based on CUDA

In this study, the parallel computing program for multi-body system dynamics is developed using the CUDA C language provided by NVIDIA. As the device (GPU) has more memory than the host (CPU), understanding device memory features is important to use it efficiently. To use the device memory, it should be allocated from the host. Afterwards, the memory on the device should be freed. If the device memory needs host data, the host data should be given using the CUDA API function. The CUDA API offers a useful function to parallel programming users [3]. In parallel programming, the data transfer between the host and the device should be minimized because such transfer has lower bandwidths compared with data transfers between global memory and device.

In this study, the host calculates the initial system data (t_0), transfers the initial data (t_0) to the device memory, and deallocates memory if simulation time is done. The device executes the integrator and transfers the solution of the system at each time step.

The flow diagram for parallel programming is shown in Fig. 4. In the host, the position and velocity vector is initialized, and acceleration can be obtained by solving the equations of motions at the initial time. These vectors are transferred to the GPU memory. In the GPU, all the positions and velocities are obtained at each time step using the HHT integrator. If the obtained solutions are acceptable, the solutions are saved to the CPU memory, and simulation time is updated. If the updated time is the designated end time, the CPU frees memory on the GPU and stops the simulation.

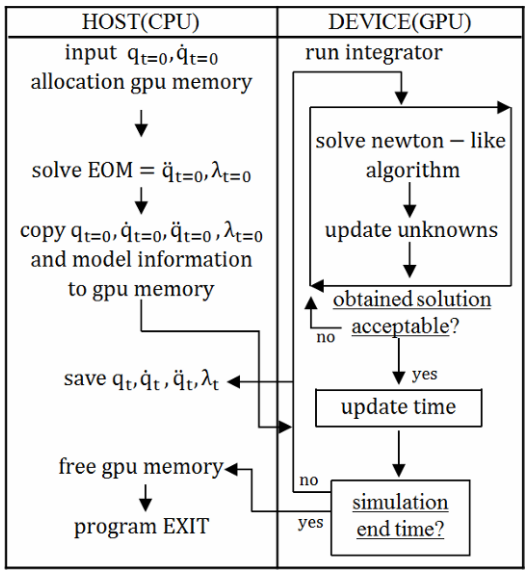

Fig. 4. Parallel programming flow diagram.

Fig. 5 shows a parallel programming flow diagram of the collision analysis between a multi-body system and numerous particles. "serial computing" or "parallel computing" in the first column of Fig. 5 means that the data that must be calculated should be processed using the CPU or GPU, as shown in Fig. 5.

GPU computing is performed by the kernel function that is executed on the GPU by the function type qualifier "__global__." When the kernel function is called, the number of threads per block and the number of blocks per grid specified in the <<< Dg, Db, Ns >>> syntax can be of type int or dim3 between the function name and the parenthesized argument list.

In a kernel function, a grid with blocks and threads is generated (Fig. 6). Each block within the grid can be identified by a one-dimensional, two-dimensional, or three-dimensional index accessible within the kernel through the built-in blockIdx variable. Each thread that executes the kernel is given a unique thread ID that is accessible within the kernel through the built-in threadIdx variable. The thread block dimension is accessible within the kernel through the built-in blockDim variable.

Parallel computing is possible by simultaneously executing the kernel function routine in each thread. If the kernel function is executed, each thread obtains the element data for calculation. Subsequently, parallel computing is carried out because a thread has the iteration structure data in the sequential program.

5. Simulation and results

To study the GPU application effects on the collision analysis of a constrained multi-body system and numerous particles, a CPU-based analysis program (CMAP) and a GPU-based

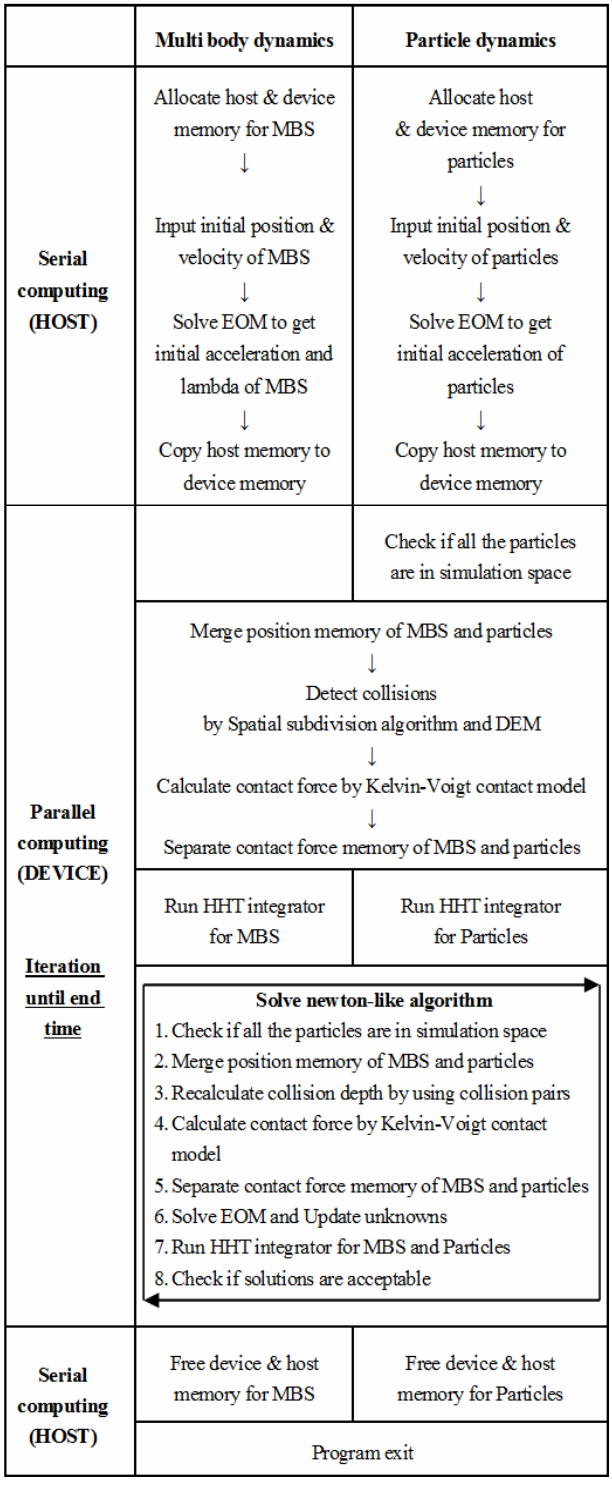

Fig. 5. Parallel programming flow diagram of the collision analysis of a multi-body system and numerous particles.

analysis program (GMAP) are respectively developed. A CMAP has numerous serial iteration statements, and a GMAP computes most parallel numerical data. In this study, ADAMS is used to verify the results obtained from the GMAP.

Table 1. Property of a multi-body system model and the simulation condition.

Attribute	Values
Mass of body	1 (kg)
Inertia of body	0.005 (kg-m^2)
Radius of body	0.05 (m)
Stiffness coefficient of spring-damper force	1.0×10^5 (N/m)
Damping coefficient of spring-damper force	100 (N-s/m)
Simulation time	1 (sec)
Time step	0.001 (sec)

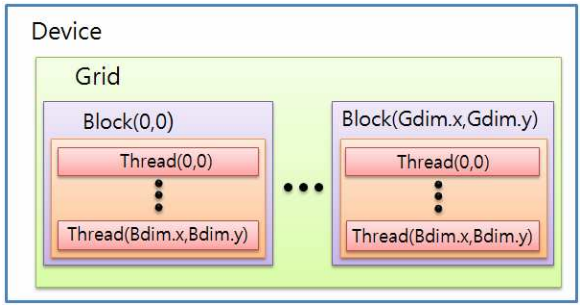

Fig. 6. Grid generation with blocks and threads.

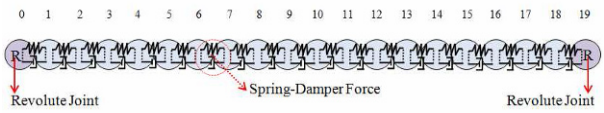

Fig. 7. A multi-body system model for GMAP verification.

5.1 GMAP verification

A multi-body system model for GMAP verification is designed for this study. An example is composed of 20 circle-shaped bodies, 19 spring-damper force elements, and 2 revolute joints (Fig. 7). In Fig. 7, each number represents the body number, wherein bodies 0 and 19 are constrained in the ground by revolute joints. Gravity is only applied as an external force, and gravitational acceleration is set at 9.8066 (m/sec^2). The properties of this model and the simulation condition are shown in Table 1.

Figs. 8 and 9 show the horizontal and vertical displacements of bodies 4 and 5, whose initial positions are (-0.55, -0.6) and (-0.45, -0.60), respectively. As shown in Figs. 8 and 9, the GMAP results are identical to the ADAMS results.

5.2 Collision simulation

A collision model for GMAP contact force verification is developed. The model is composed of 18 circle-shaped bodies and 13 revolute joints (Fig. 10). In Fig. 10, each number represents the body number. 13 bodies (from 0 to 12) are constrained by revolute joints, and 5 bodies (from 13 to 17) are

Table 2. Properties of a collision model and the simulation condition.

Attribute	Values
Mass of body	1 (kg)
Inertia of body	0.005 (kg-m^2)
Radius of body	0.05 (m)
Stiffness coefficient of contact force	1.0×10^7 (N/m)
Damping coefficient of contact force	10 (N-s/m)
Simulation time	1 (sec)
Time step	0.001 (sec)

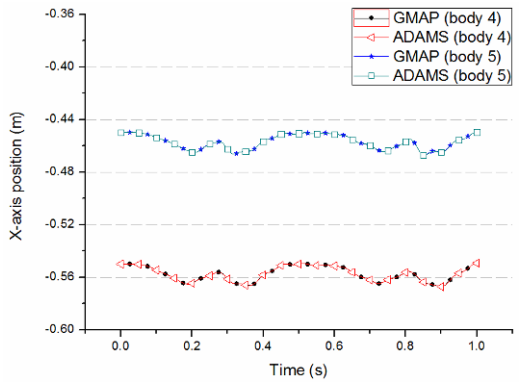

Fig. 8. Horizontal displacement comparison of body 4.

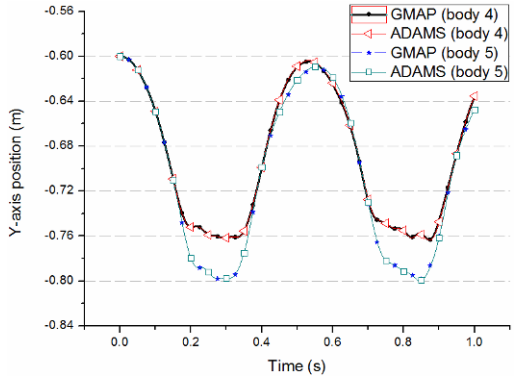

Fig. 9. Vertical displacement comparison of body 4.

considered as particles without constraints. The properties of this model and the simulation condition are shown in Table 2.

Figs. 11 and 12 show the horizontal and vertical displacement of particles 13 and 14. The initial position of particle 13, which is (-0.60, 1.50), is in contact with body 3, whose position is (-0.65, -0.90). The initial position of particle 14, which is (-0.40, 1.30), is in contact with body 5, whose initial position is (-0.45, -1.10).

From Figs. 11 and 12, the results are the same as the ADAMS results.

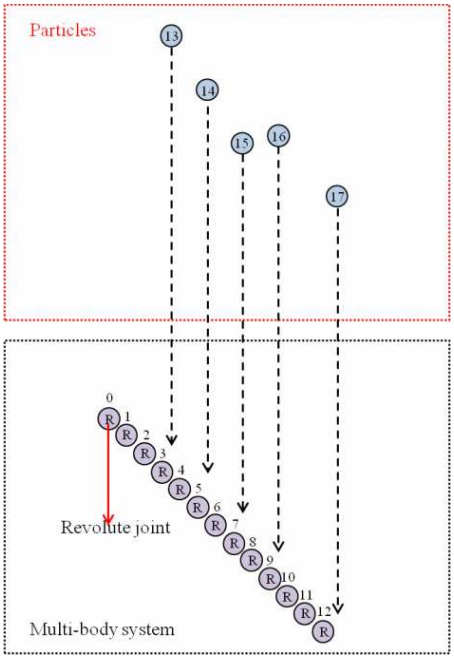

Fig. 10. A collision model for GMAP contact force verification.

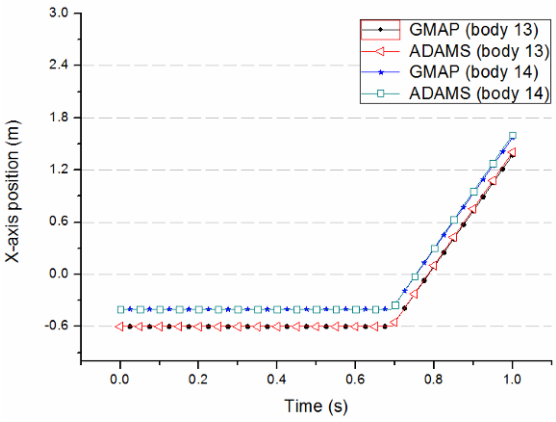

Fig. 11. Horizontal displacement comparison of particle 13.

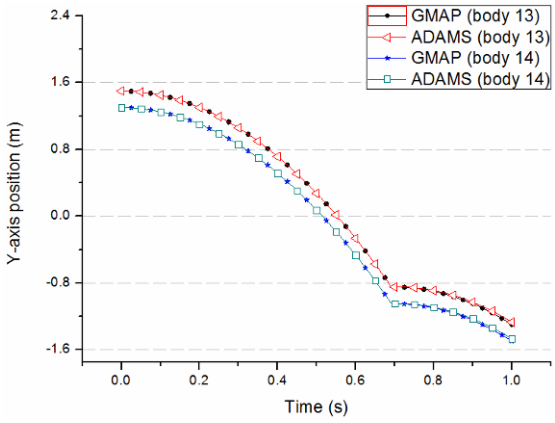

Fig. 12. Vertical displacement comparison of particle 13.

Table 3. Properties of a multi-body system and the simulation condition for the particle collision analysis.

Attribute	Values
Mass of body	1 (kg)
Inertia of body	0.005 (kg-m^2)
Radius of body	0.05 (cm)
Stiffness coefficient of spring-damper force	1.0×10^5 (N/m)
Damping coefficient of spring-damper force	100 (N-s/m)
Stiffness coefficient of contact force	1.5×10^6 (N/m)
Damping coefficient of contact force	10 (N-s/m)
Simulation time	1 (sec)
Time step	0.001 (sec)

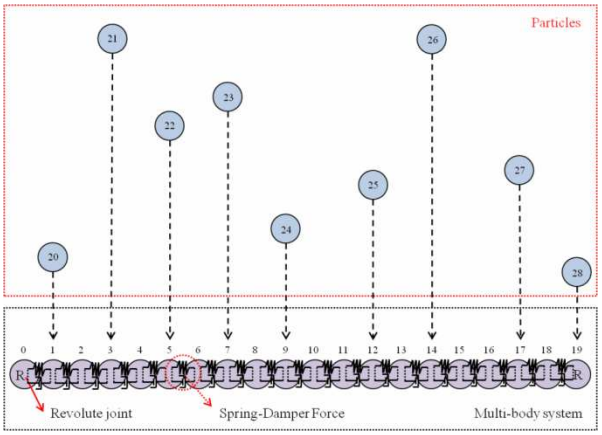

Fig. 13. Collision simulation model between a multi-body system and numerous particles.

5.3 Collision simulation between multi-body model and particles

Fig. 13 shows the model for collision analysis between a multi-body system and numerous particles. The model is composed of 29 bodies, 19 spring-damper force elements, and 2 revolute joints. All bodies are considered circular, rigid bodies so that the DEM can be applied to all bodies. Rigid bodies are connected by spring dampers. 9 bodies (from 20 to 28) in the top box are particles without constraints. The properties of this model and the simulation condition are shown in Table 3. Fig. 14 illustrates the simulation when 400 particles collide with a multi-body system.

Fig. 15 shows the comparison of computing time according to the number of particles. In Fig. 15, the circular symbol line represents the results obtained from the GMAP. The accuracy of the GMAP with regard to the number of particles is verified by comparing it with that of the CMAP. The GMAP provides an accurate calculation regardless of the number of particles. Notably, the greater the number of particles, the more comput-

Table 4. Computing time comparison between GMAP and CMAP according to the number of particles.

Particle number (N)	GMAP	CMAP
100	24.0	20.3
150	24.4	23.4
200	24.7	26.9
250	25.2	30.4
300	25.8	34.4
350	26.2	37.9
400	26.9	42.3
450	27.4	48.1
500	27.8	54.8
550	28.6	63.2
600	29.2	70.9
650	29.9	81.1
700	30.4	93.8
750	30.8	104.8
800	31.2	116.5
850	31.5	129.6
900	31.8	150.1

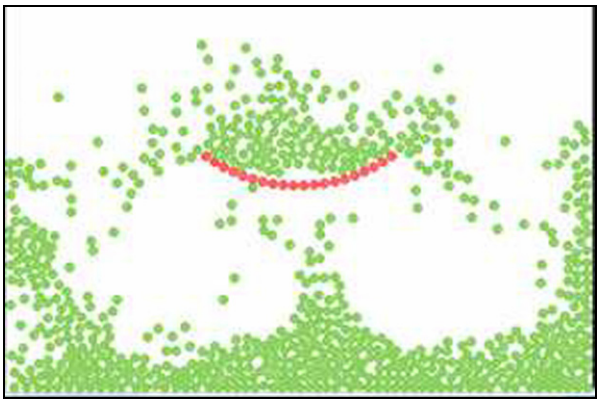

Fig. 14. Collision simulation of a multi-body system and 400 particles.

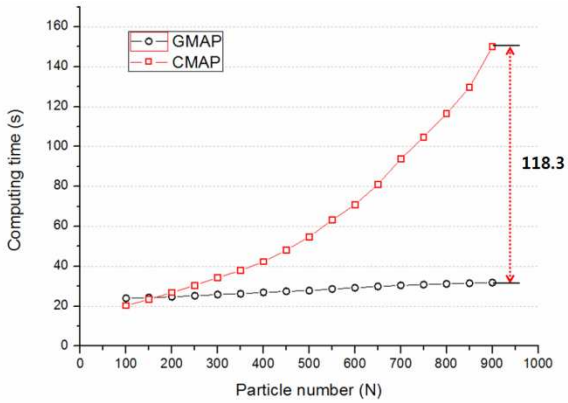

Fig. 15. Computing time comparison between GMAP and CMAP according to the number of particles.

ing time can be saved. As shown in Table 4, when the number of particles is 900, the computing speed of the GMAP is about five times faster than that of the CMAP.

This result can be attributed to the fact that whereas the CMAP computes the positions of particles one after another with one thread, the GMAP calculates the positions of all particles at the same time using many threads.

6. Conclusions

In this study, we proposed a parallel algorithm using a GPU and explained the data interface of the CPU and GPU. A collision analysis model, which consists of a multi-body system and numerous particles, was constructed to compare the numerical efficiency of parallel programming with that of serial programming. A motion equation was formulated using the absolute coordinate system, and the implicit HHT integration algorithm was used for the integral equation. C and C++ were used to program the multi-body dynamics analysis, and ADAMS was used to test solution accuracy.

The spatial subdivision algorithm and the DEM are very effective for collision detection. However, numerous particles remain in the simulation space. Notably, the greater the number of particles, the more computing time can be saved. The calculating time of the serial program increased in proportion to the number of particles. For the multi-body dynamic analysis with numerous contact bodies, CUDA-based parallel programming is strongly recommended.

Acknowledgment

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea funded by the Ministry of Education, Science, and Technology (2011-0011845).

References

[1] *NVIDIA CUDA™*, NVIDIA CUDA C Programming Guide, Version 4.1, 2011-11-18.

[2] *NVIDIA CUDA™*, CUDA Toolkit 4.1 CUBLAS Library, PG-05326-041_v01, October (2011).

[3] *EM Photonics*, Inc., CULA Reference Manual, Release R13(CUDA4.0), November 08 (2011).

[4] J. Sanders and E. Kandrot, CUDA by examples : An introduction to general-purpose GPU programming, *Addison Wesley* (2011).

[5] D. Negrut, A. Tasora, M. Anitescu, H. Mazhar, T. Heyn and A. Pazouki, Solving large multibody dynamics problems on the GPU, *Math, and Comp. Science Division*, ANL/MCS-P1777-0710 (2010).

[6] A. Tasora, D. Negrut and M. Anitescu, GPU-based parallel computing for the simulation of complex multibody systems with unilateral and bilateral constraints: An overview.

[7] S. L. Grand, Broad-phase collision detection with CUDA, GPU GEMS 3, *Addison Wesley* (2007).

[8] S. Green, Particle simulation using CUDA, *NVIDIA Corporation* (2010).

[9] M. Tupy, A study on the dynamics of granular material with a comparison of DVI and DEM approaches, *University of Wisconsin-madison* (2010).

[10] P. E. Nikravesh, *Computter-aided analysis of mechanical systems*, Prentice-Hall International. Inc.

[11] A. A. Shabana, *Computational dynamics*, 2 Ed. Wiley-Interscience (2001).

[12] D. Negrut, R. Rampalli, G.Ottarsson and A. Sajdak, On the use of the HHT method in the context of index 3 differencial algebraic equations of multibody dynamics, *International Journal for Numerical Methods in Engineering* (2000) 00:1-6.

[13] D. Negrut, G. Ottarsson, R. Rampalli and A. Sajdak, On an implementation of the Hilber-Hughes-Taylor method in the context of index 3 differential-algebraic equations of multibody dynamics, *International Journal for Numerical Methods in Engineering* (2002).

[14] N. Schafer and D. Negrut, On the potential of implicit integration methods for molecular dynamics simulation, *Journal of Computational Physics* (2009).

[15] *MSC Software ADAMS help*, MSC Software.

Jeong-Hyun Sohn has been an associate professor in the Department of Mechanical and Automotive Engineering of Pukyong National University since 2003. His interests are multi-body system dynamics, mechanism design, and dynamic analysis of vehicle system.

Chul-Woong Jun is a graduate student of mechatronics engineering at Pukyong National University. His interests are parallel computing, high-performance computing, and multi-body system dynamics.

Hye-young Jung is a graduate student of mechatronics engineering at Pukyong National University. Her interests are kinematic design and dynamic analysis of mechanical systems.