Performance improvements of differential operators code for MPS method on GPU

Kohei Murotani¹ · Issei Masaie² · Takuya Matsunaga¹ · Seiichi Koshizuka¹ ·
Ryuji Shioya³ · Masao Ogino⁴ · Toshimitsu Fujisawa²

Received: 14 January 2015 / Revised: 15 May 2015 / Accepted: 8 July 2015 / Published online: 21 July 2015
© OWZ 2015

Abstract In the present study, performance improvements of the particle search and particle interaction calculation steps constituting the performance bottleneck in the moving particle simulation method are achieved by developing GPU-compatible algorithms for many core processor architectures. In the improvements of particle search, bucket loops of the cell-linked list are changed to a loop structure having fewer local variables and the linked list and the forward star of particle search algorithms within a bucket are compared. In the particle interaction calculation, the problem of the ratio of particles within the interaction domain to the neighboring particle candidates being quite low is improved. By these improvements, a performance efficiency of 24.7 % can be achieved for the first-order polynomial approximation scheme using NVIDIA Tesla K20, CUDA-6.5, and double-precision floating-point operations.

Keywords MPS method · CUDA · Gradient · Laplacian · Differential operators

1 Introduction

The moving particle simulation (MPS) method [1,2] is a popular particle method along with the smoothed particle hydrodynamics (SPH) method [3,4]. The particle method regards a continuum as a set of particles, discretizes the physi-

cal laws governed by differential equations using interactions between the particles, and calculates the states and motions of the particles. Since the particles, as the calculation points, move at discrete time steps, the particle method is superior to grid methods in terms of solving dynamic physical phenomena such as free surfaces and large deformations. However, in general, fluid analysis by the particle method, only particle data within the interaction domain are referred to and the reference relationship can change at each step. This property of the particle method makes it difficult to achieve a high performance efficiency in recent, advanced parallel computers.

On distributed parallel computer, realistic large-scale analysis using several hundred million particles has been already realized [5–8]. On the other hand, research on particle simulation using GPUs is becoming more frequent as the performance of GPUs increases. Initially, parallel computing algorithms used graphic API [9,10]. Today, by programming environments such as CUDA that have been developed by hardware vendors, research on GPUs has more common.

Algorithms and optimization methods described in NVIDIA Documentation [11] of the target problems utilized in the present paper are commonly used in recent studies of particle methods. Particle simulations using GPUs include studies on the n-body problem [12,13], molecular dynamics [14–16], cloth and fluid simulation [17], rigid body simulation [18], powder simulation by the discrete element method [19–21], and computational fluid dynamics by the MPS and the SPH methods [22–29], involving various fields. In the field of the computational fluid dynamics by the MPS and the SPH methods, although relative evaluations in terms of parallel scalability are frequently performed, absolute evaluations (quantitative evaluation) of performance efficiency are also required. One objective of the present study is to develop algorithms with a high performance efficiency in which high GPU utilization is achieved.

✉ Kohei Murotani
muro@mps.q.t.u-tokyo.ac.jp

¹ Graduate School of Engineering, The University of Tokyo,
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan

² Prometech Software, Inc., Tokyo, Japan

³ Toyo University, Tokyo, Japan

⁴ Nagoya University, Nagoya, Japan

In the MPS method, the particle interaction within the interaction domain is calculated. When particles within the interaction domain are searched, the distance between particles is calculated and the distance is compared with the radius of the interaction domain. If a particle is compared with all other particles, then the calculation becomes very large, because (the number of particles) $\times$ (the number of particles) comparisons are required. However, in the MPS method, since only particles within the interaction domain are calculated, if a narrowing-down processing to search particles is performed in advance, then the comparisons between all particles are not required. In the present study, cell-linked lists [11, 21, 30–37] of a high-speed algorithm to search all particle pairs within the interaction domain in molecular dynamics simulations are adopted.

In the cell-linked lists, the bounding box of the entire analysis domain is filled with uniform buckets of cubes, and all particles are assigned to corresponding buckets such that each bucket has a sequence of particle data assigned to it as linked list. In the present study, only the particle interaction calculation is described. Ideally, all aspects of the calculation process including the generation of the cell-linked lists should be estimated, as performed in many related studies [16, 21, 35, 38, 39]. However, since the generation of the cell-linked lists is essentially a sort, studies of the radix sort [35–37] for GPUs have already been performed. Additionally, in the MPS method, since the generation of the cell-linked lists is performed once per time step and at least four particle interaction calculations are performed in each time step, the generation of the cell-linked lists is small in comparison to the particle interaction calculations in the case of the MPS method [5]. Therefore, in the present paper, only the particle interaction calculations, which constitute the majority of the overall calculation, are discussed.

In Sect. 2, the particle interaction models of the MPS method are presented. In Sect. 3, performance improvements are described. In Sect. 3.1, particle search algorithms such as the cell-linked lists, the linked list and the forward star are described. In Sect. 3.2 and 3.3, improvements for steps (sub-methods) of the MPS method having low performance efficiency are presented. In Sect. 3.4, the obtained performance efficiencies for the proposed improved methods are presented.

2 MPS method

2.1 MPS method

The MPS method developed by Koshizuka and Oka [1, 2] is a numerical calculation method which regards a continuum as a set of particles, discretizes the physical laws governed by differential equations using interactions between parti-

cles, and calculates the states and motions of the particles by Lagrangian particle tracking. The governing equations are the equation of state and Navier–Stokes equations for an incompressible flow, which are expressed as

$$\frac{D\rho}{Dt} = 0, \quad (1)$$

$$\frac{D\mathbf{u}}{Dt} = -\frac{1}{\rho}\nabla P + \nu\nabla^2\mathbf{u} + \mathbf{F}, \quad (2)$$

where ρ , t , $\mathbf{u}$, P , ν , and $\mathbf{F}$ are density, time, flow velocity vector, pressure, kinematic viscosity, and external force, respectively.

In the MPS method, the governing Eqs. (1) and (2) are discretized by replacing the differential operators with approximating discrete operators based on a particle interaction model. The gradient model is defined as

$$\langle\nabla\phi\rangle_i = \frac{d}{n^0} \sum_{j \neq i} \left[w(|\mathbf{r}_{ij}|) \frac{\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|} \frac{\phi_{ij}}{|\mathbf{r}_{ij}|} \right], \quad (3)$$

and the Laplacian model is defined as

$$\langle\nabla^2\phi\rangle_i = \frac{2d}{\lambda^0 n^0} \sum_{j \neq i} \left[w(|\mathbf{r}_{ij}|) \phi_{ij} \right], \quad (4)$$

where d , $\mathbf{r}_i$, ϕ_i , $w(|\mathbf{r}_{ij}|)$ are the number of space dimensions, position of particle “ i ,” scalar value of particle “ i ,” and weight function, respectively, $\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i$ and $\phi_{ij} = \phi_j - \phi_i$. The radius of the interaction domain r_e is defined as

$$r_e = c_0 \times l_0, \quad (5)$$

where l_0 and c_0 are an initial spacing between particles and a coefficient of the radius of the interaction domain, respectively. Either 2.1 or 3.1 is usually adopted as the value of c_0 . The term n^0 , which is a normalization parameter of the gradient model (3), is defined as

$$n^0 = \sum_{j \neq i} w(|\mathbf{r}_{ij}|). \quad (6)$$

λ^0 , which is a normalization parameter of the Laplacian model (4), is defined as

$$\lambda^0 = \frac{\sum_{j \neq i} w(|\mathbf{r}_{ij}|) |\mathbf{r}_{ij}|^2}{\sum_{j \neq i} w(|\mathbf{r}_{ij}|)}. \quad (7)$$

2.2 Higher order spatial derivative scheme

The particles interaction models (3) and (4) of the MPS method assume that particles within the interaction domain

are regularly located in a grid. However, since this assumption is not satisfied for a random distribution of particles including boundaries, the computational accuracy is low, as pointed out by Tamai and Koshizuka [40]. A higher order spatial derivative schemes such as the moving least squares particle hydrodynamics method (MLSPH) [41] and the reproducing kernel particle methods (RKPM) [42] have been proposed to avoid this assumption. In the present study, the higher order spatial derivative scheme of the least squares moving particle semi-implicit (LSMPS) method based on the Taylor expansion proposed by Tamai and Koshizuka [40] is adopted. By omitting coefficient of the Taylor expansion, the higher order spatial derivative scheme of LSMPS [15] can be simplified to

$$\mathbf{M}_i = \sum_{j \neq i} \left[w(|\mathbf{r}_{ij}|) \mathbf{p}(\mathbf{r}_{ij}^*) \otimes \mathbf{p}(\mathbf{r}_{ij}^*) \right], \quad (8)$$

$$\mathbf{b}_i = \sum_{j \neq i} \left[w(|\mathbf{r}_{ij}|) \phi_{ij} \mathbf{p}(\mathbf{r}_{ij}^*) \right], \quad (9)$$

$$\mathbf{x}_i = \mathbf{M}_i^{-1} \mathbf{b}_i, \quad (10)$$

where $\mathbf{p}(\mathbf{r}_{ij}^*)$ is a single-term vector generated by $\mathbf{r}_{ij}^*$, $\mathbf{r}_{ij}^* = \mathbf{r}_{ij}/r_s$, and r_s is scaling parameter. Calculation methods adopting higher order spatial derivative schemes such as MLSPH [41] and RKPM [42] include equations of the form given in Eqs. (8), (9), and (10).

Calculation targets in the present study are three models: the gradient model (3), the Laplacian model (4), and the first-order polynomial approximation scheme of the higher order spatial derivative scheme defined by Eq. (10).

3 Performance improvements

3.1 Particle search algorithms

3.1.1 Cell-linked lists

In this subsection, the cell-linked lists of a high-speed algorithm to search particles are described [11, 21, 30–34]. First, after a bounding box of the whole analysis domain is defined, the bounding box is filled with uniform buckets of cubes as shown in Fig. 1a. In the present study, the analysis domain is extended from the boundaries by one bucket width in order to remove the exception handlings which occur in the buckets at the boundary. It means that particles do not exist on extended buckets. If the bucket width is larger than the radius of the interaction domain, then when particles within the interaction domain are searched, only particles belonging to the $3 \times 3 \times 3$ buckets centered on the bucket to which the target particle belongs need be searched. The bucket width is defined as

$$l_b = c_b \times r_e \quad (c_b \geq 1.0), \quad (11)$$

where c_b is a coefficient of bucket width determined by particle movement. Figure 1 shows the relationship between particles and buckets in the case of $c_0 = 2.0$, $c_b = 1.0$.

As shown in Fig. 1b, a bucket number is defined as

$$ib = ix + nBx \times (iy + nBy \times iz), \quad (12)$$

where nBx , nBy are the number of buckets in the x , y axial directions, and ix , iy , iz are the bucket coordinates of x , y , z axial directions. The bucket coordinates ix , iy , iz are calculated as

$$\begin{aligned} ix &= \lfloor (r_x^a - \text{MIN_X}) * \text{DBinv} \rfloor, \\ iy &= \lfloor (r_y^a - \text{MIN_Y}) * \text{DBinv} \rfloor, \\ iz &= \lfloor (r_z^a - \text{MIN_Z}) * \text{DBinv} \rfloor, \end{aligned} \quad (13)$$

where r_x^a , r_y^a , r_z^a are the x , y , z coordinates of a particle “a,” DBinv is the inverse of the bucket width, and MIN_X , MIN_Y , MIN_Z are the origin coordinates of the analysis domain.

To search the neighboring particles of the particle “a” in Fig. 1c, ix , iy , iz are calculated using Eq. (13). Next, the 27 neighboring buckets in Fig. 1c are obtained by Eq. (12) as shown in Program 1. If all the particles belonging to the 27 neighboring buckets are found, particles within the interaction domain can be obtained by comparison with distance from the particle “a” as shown in Fig. 1d. The method to search particles belonging to a bucket will be described in Sects. 3.1.4 and 3.1.5.

Program 1 Calculation method of 27 neighboring buckets

```

1  for (jz = iz-1; jz <= iz+1; jz++) {
2  for (jy = iy-1; jy <= iy+1; jy++) {
3  for (jx = ix-1; jx <= ix+1; jx++) {
4      jb = jx + nBx * (jy + nBy * jz);
5  }}}

```

3.1.2 Simplification of neighboring buckets calculation

In this subsection, a method to simplify the neighboring buckets calculation of Program 1 is described. First, a neighboring bucket number offset array `bucket_offsets[27]` is prepared in advance as shown in Program 2. `bucket_offsets[27]` generated from only nBx , nBy stores the differences between a bucket number “ ib ” and the 27 neighboring buckets. In the case of CUDA, if `bucket_offsets[27]` is stored on the constant memory as lines 11, 12 of Program 2, then high-speed access can be performed by using the read-only cache.

Fig. 1 Relations of particles and buckets. **a** Assigning particles to buckets. **b** Bucket numbers. **c** Particle “a” and buckets within interaction domain. **d** Particle “a” and particles within interaction domain

Program 2 How to generate a neighboring bucket number offset array

```

1  __constant__ int d_bucket_offsets[27];
2  int h_bucket_offsets[27];
3
4  k=0;
5  for (jz = -1; jz <= 1; jz++){
6  for (jy = -1; jy <= 1; jy++){
7  for (jx = -1; jx <= 1; jx++){
8  h_bucket_offsets[k++]
9  =jx + nBx * (jy + nBy * jz);
10 }
11 }
12 cudaMemcpyToSymbol(d_bucket_offsets,
13 h_bucket_offsets, sizeof(int) * 27);

```

If `bucket_offsets[27]` is used, Program 1 can be changed to Program 3. If they are compiled under version 6.5 of the CUDA environment, three registers per thread are consumed for a for-loop on lines 1, 2, 3 of Program 1 and two registers per thread are consumed for a for-loop on line

Program 3 How to use a neighboring bucket number offset array

```

1  for (io = 0; io < 27; io++) {
2  jb = ib + bucket_offsets[io];
3  }

```

1 of Program 3. Therefore, if the three for-loops of Program 1 can be fused to the for-loop of Program 3, then the number of registers per thread can be reduced from nine to two. Next, since `nBx`, `nBy` on line 4 of Program 1 disappears and `ib` on line 2 of Program 3 appears, one register per thread can be removed. In all, eight registers per thread can be removed. Additionally, integer operations on line 4 of Program 1 disappear.

In the case of a Kepler architecture, the number of registers per thread must be less than 32, for theoretical occupancy achieves 100 %. It is important to reduce the number of registers per thread by simplifying the neighboring buckets calculation, because this improves multiprocessor (SMX) utilization.

Fig. 2 How to access neighboring buckets. **a** Access in units of one bucket. **b** Access in units of three buckets

3.1.3 Method to access neighboring buckets in units of three

The method to access neighboring buckets in units of three [29] is adopted in this research. Figure 2a shows how to access neighboring buckets in units of one or three buckets. As shown, `bucket_offsets[27]` of Program 3 can be accessed for all 27 neighboring buckets. If the forward star structure described in Sect. 3.1.5 is used, since particle data are continuously arranged in the x bucket coordinate direction, `bucket_offsets[27]` of Program 4 can be accessed in units of three buckets as shown in Fig. 2b. Therefore, since access in units of one bucket can be improved to access in units of three buckets, the efficiency of particle data transfer can be increased. However, this method cannot be used for the linked list structure described in Sect. 3.1.4.

Program 4 Method to access neighboring buckets in units of three

```
1  for (io = 0; io < 27; io+=3) {
2      jb = ib + bucket_offsets[io];
3  }
```

3.1.4 Search of particles by a linked list

When cell-linked lists are used, all particles must be assigned to corresponding buckets. Conversely, each bucket must have a sequence of particle data assigned to it in some way. Specifically, a sequence of particle numbers in a linked list and the particle data themselves in the forward star are assigned to each bucket. The linked list [19] in this subsection and the forward star [11, 21, 43–52] in next subsection are described.

For example, if six particles are assigned into four buckets using a standard one-way linked list as shown in Fig. 3a, then the two arrays `FirstIdxInBucket[4]={0, 4, -1, 1}` and `NextIdx[6]={2, 3, -1, 5, -1, -1}` must be prepared. In the array `FirstIdxInBucket[]`, the first particle number in each bucket is written. If the component of `FirstIdxInBucket[]` is “-1,” then the bucket is empty. In the array `NextIdx[]`, the particle num-

Fig. 3 Particle data access pattern by linked list and forward star. **a** Linked list. **b** Forward star

ber linked to a next particle is written. If the component of `NextIdx[]` is “-1,” then no more particles exist in the bucket. In the linked list, read in memories become “indirect access” in order to be accessed through `NextIdx[]`.

Program 5 is a program to search particles assigned to a bucket by the linked list. `jb` is a bucket number. `jp` is a particle number. On line 1, the first particle number `jp` assigned to a bucket `jb` is called. On line 2, if `jp=-1`, since the bucket is empty, `continue` skips the procedures in the for-loop. On line 5, `jp` is updated to a next particle number. If `jp` has become “-1,” since no more particles exist in the bucket, `break` escapes the for-loop of the bucket.

Program 5 Search of particles assigned to bucket “jb” by linked list

```
1  jp = FirstParticleInBucket[jb];
2  if (jp == -1) continue;
3  for (;;) {
4      Calculation of a particle 'jp.'
5      jp = NextParticleInBucket[jp];
6      if (jp == -1) break;
7  }
```

3.1.5 Search of particles by forward star

In the forward star [11, 21], as shown in Fig. 3b, particle data can be rearranged or sorted by buckets after particle numbers are assigned to each bucket. The particle data need not be sorted within each bucket. If the boundaries of particle numbers between buckets are prepared, the particle numbers need not be assigned to each bucket. In the present study, a radix sort [35–37] which regards bucket numbers as keys is performed in order to sort particle data. For example, as shown in Fig. 3b, if particle data are rearranged by buckets, the boundaries of particle numbers between buckets can be prepared as an array of `NumParticleInBucket={0, 2, 3, 3, 6}`.

Program 6 is a program to search particles assigned to a bucket by the forward star. The particle data in the bucket `jb` exist from `CountParticleInBucket[jb]`

to `CountParticleInBucket[jb+1]-1`. The read pattern of particle data in Program 6 is the same as the read pattern of column indices in compressed row storage (CRS). In the forward star, read in memories become “direct access” in order to directly access particle data.

Program 6 Search of particles assigned to bucket “jb” by forward star

```

1  for (jp=CountParticleInBucket[jb];
2      jp<CountParticleInBucket[jb+1];
3      jp++)
4  {
5      Calculation of a particle ``jp``
6  }
```

3.2 Bucket width

In the MPS method, the largest cause of a low performance efficiency lies in a quite low ratio of particles within the spherical interaction domain to neighboring particle candidates. The mean number of neighboring particle candidates is $(3l_b)^3$, the volume of a cube $3l_b$ on a side. The mean number of particles within a spherical interaction domain is $\frac{4}{3}\pi r_e^3$, and the volume of a sphere of radius r_e . Therefore, the ratio of particles within the spherical interaction domain to neighboring particle candidates becomes, by Eq. (5),

$$\frac{\frac{4}{3}\pi r_e^3}{(3l_b)^3} = \frac{\frac{4}{3}\pi}{(3c_b)^3}. \quad (14)$$

For example, when $c_b = 1.0$, Eq. (14) gives 0.155. If discretized, Eq. (14) tends to give a larger value.

As shown in Fig. 4, if the bucket width becomes smaller, the number of neighboring particle candidates becomes generally fewer. Although bucket width l_b as defined in Eq. (11) is used in standard cell-linked lists,

$$l_b = c_b r_e / n_{\text{div}} \quad (15)$$

is adopted in the present study, where n_{div} is the number of divisions for a bucket. Dividing the buckets causes the number of neighboring buckets to become $(1 + 2n_{\text{div}})^3$ from 27. Figure 4a shows an example with $n_{\text{div}} = 1$, whereas Fig. 4b shows an example with $n_{\text{div}} = 3$. Dotted circles in Fig. 4b indicate particles which were removed from being neighboring particle candidates.

One merit of this method is that, as n_{div} increases, the ratio of particles within the interaction domain to neighboring particle candidates increases by decreasing the number of neighboring particle candidates. On the other hand, one demerit of this method is that, since the number of buckets increases by a factor of $(n_{\text{div}})^3$, the amount of memory required by the arrays, which depends on the number of

Fig. 4 Example of changes in neighboring particle candidates for changes in bucket width. **a** $n_{\text{div}} = 1$. **b** $n_{\text{div}} = 3$

Fig. 5 Example of changes in the particles within the interaction domain for a change in the interaction domain shape. **a** Spherical interaction domain. **b** Cubic interaction domain

buckets, increases. In Sect. 3.4.4, the computation time and performance efficiency are shown for different bucket widths.

The values of the spatial derivative models (3), (4), and (10) remain unchanged by introducing the modifications of the bucket width described in this subsection.

3.3 Interaction domain shape

In this section, as shown in Fig. 5, a method giving a higher ratio of particles within the interaction domain to neighboring particle candidates by changing the interaction domain shape from spherical to cubic is presented. In most implementations of the MPS method, a spherical interaction domain as shown in Fig. 5a is used. Another cause of a low ratio of particles within the interaction domain to neighboring particle candidates is the use of a spherical interaction domain in spite of the buckets being cubes.

In the present study,

$$\left(\frac{\|\mathbf{r}_{ij}\|}{r_e} - 1 \right)^2 \quad (16)$$

is adopted as a weight function for a spherical interaction domain, and

$$\left(\frac{|x_{ij}|}{r_e} - 1\right)^2 \times \left(\frac{|y_{ij}|}{r_e} - 1\right)^2 \times \left(\frac{|z_{ij}|}{r_e} - 1\right)^2 \quad (17)$$

is adopted as a tensor product weight function for a cubic interaction domain. x_i, y_i, z_i are the x, y, z coordinates of a particle “ i ,” $x_{ij} = x_j - x_i, y_{ij} = y_j - y_i, z_{ij} = z_j - z_i$, and $\|\mathbf{r}_{ij}\| = \sqrt{x_{ij}^2 + y_{ij}^2 + z_{ij}^2}$. The second-order spike functions given in Eqs. (16) and (17) are adopted as weight functions in the present study because the weight function (17) does not require the calculation of square roots or divisions, which have high operation costs.

There are two merits for changing the shape of the interaction domain. The first merit is that, since the number of particles within the interaction domain increases, the calculation of the spatial derivative models becomes more stable. Alternatively, the size of the cubic interaction domain can be made small so as to enclose the same number of particles as the spherical interaction domain. The second merit is that the calculation time can be reduced, because the weight function (17) does not require square root evaluations which have large operation costs. On the other hand, one demerit is that if the size of the interaction domain is not changed, then the calculation time will increase because of increasing the number of particles within the interaction domain. In Sect. 3.4.4, the computation time and performance efficiency are shown for different interaction domain shapes.

By introducing the modifications of the interaction domain shape described in this subsection, although the values of the spatial derivative models (3), (4), and (10) are altered a little, their accuracy (approximation orders) remain unchanged.

3.4 Performances measurement

3.4.1 Measurement models and parameter setting

In this subsection, measurement models and parameter settings to measure the performance under three spatial derivative models are described. The particle model being measured is a 226,981-particle model having $61 \times 61 \times 61$ equally spaced particles. The particle data are sorted by the radix sort where the sort key is a bucket number [35–37]. For a coefficient of radius of interaction domain of $c_0 = 3.1$ and a coefficient of bucket width of $c_b = 1.1$, the number of buckets is 13,824 ($= 24 \times 24 \times 24$). The measured models are the gradient model (3), the Laplacian model (4), and the

first-order polynomial approximation scheme of

$$\begin{aligned} \begin{pmatrix} \frac{\partial \phi}{\partial x} r_s \\ \frac{\partial \phi}{\partial y} r_s \\ \frac{\partial \phi}{\partial z} r_s \end{pmatrix} &= \left[\sum_{j \neq i} w(\|\mathbf{r}_{ij}\|) \begin{pmatrix} x_{ij}^* \\ y_{ij}^* \\ z_{ij}^* \end{pmatrix} \otimes \begin{pmatrix} x_{ij}^* \\ y_{ij}^* \\ z_{ij}^* \end{pmatrix} \right]^{-1} \\ &\times \left[w(\|\mathbf{r}_{ij}\|) \phi_{ij} \begin{pmatrix} x_{ij}^* \\ y_{ij}^* \\ z_{ij}^* \end{pmatrix} \right], \end{aligned} \quad (18)$$

where $x_{ij}^* = x_{ij}/r_s, y_{ij}^* = y_{ij}/r_s, z_{ij}^* = z_{ij}/r_s$.

3.4.2 GPU-related settings

In this research, NVIDIA Tesla K20, CUDA-6.5, and NVIDIA Visual Profiler are used. In K20, the peak double-precision floating-point performance is 1.17 Tflops, the maximum number of resident warps per multiprocessor is 64, the maximum number of resident blocks per multiprocessor is 16, and the number of 32-bit registers per multiprocessor is 65,536. The Kepler architecture is used, which has an SMX multiprocessor. In this measurement, the number of threads per block (THREADS) is 128 and the number of blocks (BLOCKS) is

$$\begin{aligned} \text{BLOCKS} \\ &= (\text{TOTAL_THREADS} + \text{THREADS} - 1) / \text{THREADS}, \end{aligned} \quad (19)$$

where TOTAL_THREADS is the total number of threads.

Constant memory is used for the neighboring bucket number offset array and the material property array, and texture cache is used for read-only data.

3.4.3 Performance efficiency of linked list and forward star

Although Murotani et al. [5] confirmed that performance efficiency of forward star is higher than one of linked list in the case of CPU, performance efficiency in the case of GPU is compared in this subsection. Program 7 and Program 8 are simplified CUDA programs for searching neighboring particles by the linked list and the forward star, respectively. The method used for the linked list in Program 5 appears in lines 19, 20, 21, 34, 35 of Program 7. The method used for the forward star in Program 6 appears in lines 19, 20, 21 of Program 8. In Program 7 and Program 8, the Laplacian model (4) is calculated. The gradient model (3) and the first-order polynomial approximation scheme (18) can be calculated by changing programs after line 27 in Program 8 to

Program 7 Programing to search neighboring particles by linked list

```

1  ip=blockIdx.x*blockDim.x+threadIdx.x;
2  if (ip < NumberOfParticles
3      && ParticleType[ip] != GHOST){
4
5      pos_ix = Position_x[ip];
6      pos_iy = Position_y[ip];
7      pos_iz = Position_z[ip];
8      phi_i = Phi[ip];
9
10     ix = (int)((pos_ix - MIN_X)*DBinv);
11     iy = (int)((pos_iy - MIN_Y)*DBinv);
12     iz = (int)((pos_iz - MIN_Z)*DBinv);
13
14     lap=0.0;
15     ib = ix + nBx *(iy + nBy * iz);
16
17     for ( io = 0; io < 27; io++) {
18         jb = ib + bucket_offsets[io];
19         jp = FirstParticleInBucket[jb];
20         if(jp == -1) continue;
21         for(;;){
22             v0 = Position_x[jp] - pos_ix;
23             v1 = Position_y[jp] - pos_iy;
24             v2 = Position_z[jp] - pos_iz;
25             dist2 = v0*v0+v1*v1+v2*v2;
26
27             if(dist2 < r2
28                 && ParticleType[jp] != GHOST){
29                 dist = sqrt(dist2);
30                 w = Weight(dist, r_inv);
31                 w *= (Phi[jp] - phi_i);
32                 lap += w;
33             }
34             jp = NextParticleInBucket[jp];
35             if(jp==-1) break;
36         }
37     }
38
39     Laplacian[ip] = lap*6.0*lambda_inv;
40 }

```

Program 9 and Program 10, respectively. Position_x[], Position_y[], Position_z[], and Phi[] are arrays of the x, y, z coordinates of particles and a scalar value, respectively. Weight() comprises weight functions (16) and (17). $r_inv, rs_inv, n0_inv, lambda_inv$ are inverses of r_e, r_s, n^0, λ^0 .

Figure 6 shows performance comparisons between the linked list and the forward star. Since the number of local variables of the linked list is less than that of the forward star, the number of registers per thread of the linked list is less than that of the forward star, as shown in Fig. 6a. Therefore, the achieved occupancy of the linked list is greater than that of the forward star, as shown in Fig. 6b. Based on the comparisons in Fig. 6a and b, the performance efficiency of the linked list seems to be higher than that of the forward star, because the utilization of multiprocessors by the linked list is greater than that of the forward star. However, since the bandwidth of global stores in L1/shared memory was 281.33 MB/s in the case of the linked list and 367.91 MB/s in the

Program 8 Programing to search neighboring particles by forward star

```

1  ip=blockIdx.x*blockDim.x+threadIdx.x;
2  if (ip < NumberOfParticles
3      && ParticleType[ip] != GHOST){
4
5      pos_ix = Position_x[ip];
6      pos_iy = Position_y[ip];
7      pos_iz = Position_z[ip];
8      phi_i = Phi[ip];
9
10     ix = (int)((pos_ix - MIN_X)*DBinv);
11     iy = (int)((pos_iy - MIN_Y)*DBinv);
12     iz = (int)((pos_iz - MIN_Z)*DBinv);
13
14     lap=0.0;
15     ib = ix + nBx *(iy + nBy * iz);
16
17     for (io = 0; io < 27; io+=3) {
18         jb = ib + bucket_offsets[io];
19         for(jp=CountParticleInBucket[jb];
20             jp<CountParticleInBucket[jb + 3];
21             jp++){
22             v0 = Position_x[jp] - pos_ix;
23             v1 = Position_y[jp] - pos_iy;
24             v2 = Position_z[jp] - pos_iz;
25             dist2 = v0*v0+v1*v1+v2*v2;
26
27             if(dist2 < r2
28                 && ParticleType[jp] != GHOST){
29                 dist = sqrt(dist2);
30                 w = Weight(dist, r_inv);
31                 w *= (Phi[jp] - phi_i);
32                 lap += w;
33             }
34         }
35     }
36
37     Laplacian[ip] = lap*6.0*lambda_inv;
38 }

```

Program 9 Gradient model

```

1  .....
2  if(dist2 < r2
3      && ParticleType[jp] != GHOST){
4      dist = sqrt(dist2);
5      w = Weight(dist, r_inv);
6      w = ((ip==jp)? (0): (w/dist2));
7      w *= (Phi[jp] - phi_i);
8      grad_x += w*v0;
9      grad_y += w*v1;
10     grad_z += w*v2;
11 }
12 }
13 }
14
15 Gradient_x[ip]=grad_x*3.0*n0_inv;
16 Gradient_y[ip]=grad_y*3.0*n0_inv;
17 Gradient_z[ip]=grad_z*3.0*n0_inv;
18 }

```

case of the forward star, the utilization efficiency of L1 cache of the forward star is higher than that of the linked list. The forward star can carry out high-speed access to particle data because it allows direct access to consecutive addresses of particle data. Therefore, the computation time and the per-

Program 10 First-order polynomial approximation scheme

```

1      .....
2      if (dist2 < r2
3          && ParticleType[jp] != GHOST){
4          dist = sqrt(dist2);
5          w = Weight(dist, r_inv);
6          v0*=rs_inv;  v1*=rs_inv;
7          v2*=rs_inv;
8          M0+=v0*v0*w; M1+=v0*v1*w;
9          M2+=v0*v2*w; M3+=v1*v1*w;
10         M4+=v1*v2*w; M5+=v2*v2*w;
11         w *= (Phi[jp] - phi_i);
12         b0 += v0*w;
13         b1 += v1*w;
14         b2 += v2*w;
15     }
16 }
17 }
18
19 detM = M0*M3*M5+M2*M1*M4+M1*M4*M2
20       -M0*M4*M4-M1*M1*M5-M2*M3*M2;
21 detM_inv = 1.0/detM;
22 im0 = (M3*M5-M4*M4)*detM_inv;
23 im1 = (M2*M4-M1*M5)*detM_inv;
24 im2 = (M1*M4-M2*M3)*detM_inv;
25 im3 = (M0*M5-M2*M2)*detM_inv;
26 im4 = (M2*M1-M0*M4)*detM_inv;
27 im5 = (M0*M3-M1*M1)*detM_inv;
28 Gradient_x[ip] = (b0*im0+b1*im1+b2*im2)
29                  *rs_inv;
30 Gradient_y[ip] = (b0*im1+b1*im3+b2*im4)
31                  *rs_inv;
32 Gradient_z[ip] = (b0*im2+b1*im4+b2*im5)
33                  *rs_inv;
34 }

```

formance efficiency of the forward star become generally shorter and higher, respectively, than those of the linked list, as shown in Fig. 6c and d.

3.4.4 Performance efficiency for different bucket widths and interaction domain shapes

Table 1 shows the ratio of particles within a spherical interaction domain to neighboring particle candidates. As n_{div} increases, the ratio of particles within the interaction domain to neighboring particle candidates increases from the decrease in the number of neighboring particle candidates.

Table 2 gives the ratio of particles within a cubic interaction domain with weight function (17) to neighboring particle candidates. In Table 1, the number of particles within the interaction domain is given as 26,382,775. In contrast, in Table 2, the number of particles within the interaction domain is increased to 71,473,375 by changing the interaction domain shape from spherical to cubic. In the case of $n_{\text{div}} = 3$ shown in Table 2, the ratio is 93 %. Although the ratio in the standard MPS method is 16 %, the ratio becomes greatly increased by setting $n_{\text{div}} = 3$ and changing to a cubic interaction domain.

Figure 7 shows the computation time and the performance efficiency under different bucket widths and interaction

domain shapes. First, considering $n_{\text{div}} = 1$ versus $n_{\text{div}} = 3$, as shown in the figure, the computation time and the performance efficiency for $n_{\text{div}} = 3$ become shorter and higher, respectively, than those for $n_{\text{div}} = 1$ for both the spherical and cubic shapes. Therefore, the superior performance efficiency for a smaller bucket width makes $n_{\text{div}} = 3$ preferred to $n_{\text{div}} = 1$.

Next, the improvement from changing the shape of the interaction domain for a fixed n_{div} is discussed. As shown in Fig. 7a, the computation time of the spherical interaction domain is shorter than that of the cubic interaction domain under the same conditions. The reason for this is that the number of operations increases as the number of particles within the interaction domain increases. As shown in Fig. 7b, the performance efficiencies of the gradient model and the first-order polynomial approximation scheme increase, whereas that of the Laplacian model decreases for a change from the spherical to the cubic interaction domain. As described in Sect. 3.3, although weight function (16) for the spherical interaction domain requires square root calculations, weight function (17) for the cubic case does not. In the Laplacian model where the number of operations is already small, the performance efficiency decreases by further decreasing the number of operations. Since the gradient model and the first-order polynomial approximation scheme initially have considerable numbers of operations, the performance efficiency increases in spite of the square root operation disappearing. Although the number of particles within the interaction domain increases 2.7-fold ($=71,473,375 / 26,382,775$) by changing the interaction domain shape from spherical to cubic, the corresponding increase in computation time is less than 2.7-fold.

Finally, the first-order polynomial approximation scheme in the case of $n_{\text{div}} = 3$ and a cubic interaction domain achieving a performance efficiency of 24.7 % is discussed. The bandwidth of global stores in L1/shared memory in this calculation was 605.53 MB/s and was improved 1.64-fold compared to with $n_{\text{div}} = 1$ and a spherical interaction domain. Table 3 lists the numbers of double-precision floating-point operations for the first-order polynomial approximation scheme for the case of $n_{\text{div}} = 3$ and a spherical interaction domain. From this table, 52 % ($= 861,766,158 \times 2 / 3,318,859,727$) of double-precision floating-point operations are performed by FMA. Since the achieved occupancy of this calculation is 47.6 %, utilization of about half of the maximum multiprocessor performance ($0.52 = 24.7 \% / 47.6 \%$) is achieved.

4 Conclusions

In the present study, performance improvements for the gradient model (3), the Laplacian model (4), and the first-order

Fig. 6 Performance comparisons between linked list and forward star. **a** Registers per thread. **b** Achieved occupancy (%). **c** Computation time (ms). **d** Performance efficiency (%)

Table 1 Ratio of particles within spherical interaction domain to neighboring particle candidates versus bucket width

n_{div}	1	2	3
Number of neighboring particle candidates	160,103,007	99,252,847	76,765,625
Number of particles within interaction domain	26,382,775	26,382,775	26,382,775
Ratio (%)	16	26	34

Table 2 Ratio of particles within cubic interaction domain to neighboring particle candidates versus bucket width

n_{div}	1	2	3
Number of neighboring particle candidates	160,103,007	99,252,847	76,765,625
Number of particles within interaction domain	71,473,375	71,473,375	71,473,375
Ratio (%)	44	72	93

polynomial approximation scheme (18) are achieved. The three spatial derivative models for a 226,981-particle model are measured using NVIDIA Tesla K20 and CUDA-6.5.

First, by changing the particle search from the linked list to the forward star, the computation time of the gradient model,

Fig. 7 Performance comparison between different bucket widths and interaction domain shapes. **a** Computation time (ms). **b** Performance efficiency (%)

the Laplacian model, and the first-order polynomial approximation scheme can be decreased from 24.6, 17.2, and 23.6 ms to 17.0, 11.1, and 15.4 ms, and the performance efficiency

Table 3 Number of double-precision floating-point operations for first-order polynomial approximation scheme in the case of $n_{\text{div}} = 3$ and a cubic interaction domain

Total operations	3,318,859,727
Add operations	517,552,261
Multiply operations	1,077,775,150
Multiply accumulate operations	861,766,158

of the three models can be improved from 8.0, 8.8, and 8.9 to 11.5, 13.7, and 13.7 %, respectively. These measured results in the case of GPU are similar to the comparison between the linked list and the forward star in case of CPU by Murotani et al. [5].

Next, by changing the bucket width to one third, the computation time of the three models can be decreased to 9.3, 6.5, and 8.8 ms, and the performance efficiency of the three models can be improved to 15.1, 14.5, and 17.6 %. By changing the interaction domain shape from spherical to cubic, a performance efficiency of 24.7 % can be achieved for the first-order polynomial approximation scheme. Since the achieved occupancy of this calculation is 47.6 %, utilization of about half of the maximum multiprocessor performance ($0.52 = 24.7 \%/47.6 \%$) is achieved. From these results, enough performance improvements for the three models are achieved.

Although all parts of the calculation including the generation of the linked list and the forward star should be really estimated, this is left as future work.

Acknowledgments This research was financially supported by JSPS KAKENHI Grant Number 26390127 and JST CREST project “Development of a Numerical Library based on Hierarchical Domain Decomposition for Post Petascale Simulation.” This research was supported in part by the results of the HPCI and JHPCN Systems Research Projects (Project IDs hp120232 / hp140199 / jh130031 / 14-NA07). The authors also wish to thank Prometech Software, Inc., and all the members of the ADVENTURE project for their cooperation.

References

- Koshizuka S, Oka Y (1996) Moving-particle semi-implicit method for fragmentation of incompressible fluid. *Nucl Sci Eng (NSE)* 123:421–434
- Koshizuka S, Nobe A, Oka Y (1998) Numerical analysis of breaking waves using the moving particle semi-implicit method. *Int J Numer Methods Fluids* 26:751–769
- Lucy LB (1977) A numerical approach to the testing of the fission hypothesis. *Astron J (AJ)* 82:1013–1024
- Gingold RA, Monaghan JJ (1977) Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Mon Not R Astron Soc (MNRAS)* 181:375–389
- Murotani K, Koshizuka S, Tamai T, Shibata K, Mitsune N, Yoshimura S, Tanaka S, Hasegawa K, Nagai E, Fujisawa T (2014) Development of hierarchical domain decomposition explicit MPS method and application to large-scale tsunami analysis with floating objects. *J Adv Simul Sci Eng (JASSE)* 1(1):16–35
- Iribe T, Fujisawa T, Koshizuka S (2010) Reduction of communication in parallel computing of particle method for flow simulation of seaside areas. *Coast Eng J* 52(4):287–304
- Marrone S, Bouscasse B, Colagrossi A (2012) Numerical modeling of ship wave patterns through a hybrid OpenMP/MPI SPH solver. In: 2nd international conference on violent flows, Nantes (France), 221–228
- Lefte MD, Guilcher PM, Candelier J, Le TD, Oger G, Grenier N (2012) SPH for naval applications. In: 2nd international conference on violent flows, Nantes (France), 229–237
- Kipfer P, Segal M, Westermann R (2004) UberFlow: a GPU-based particle engine. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on graphics hardware (HWWS '04), 115–122
- Kolb A, Cuntz N (2005) Dynamic particle coupling for GPU-based fluid simulation. In Proceedings of the 18th symposium on simulation technique, pp 722–727
- Green S (2007) Particle simulation using CUDA. NVIDIA Documentation. http://docs.nvidia.com/cuda/samples/5_Simulations/particles/doc/particles.pdf. Accessed
- Wilt N (2012) CUDA handbook: N-body. Addison-Wesley, New York, pp 421–447
- Nyland L, Harris M, Prins J (2007) Fast N-body simulation with CUDA. GPU gems 3, Chapter 31. Addison-Wesley, New York
- Chen FG, Ge W, Li JH (2009) Molecular dynamics simulation of complex multiphase flow on a computer cluster with GPUs. *Sci China Ser B* 52(3):372–380
- Chen MJ, Xiao GB, Chen JX, Chun YW (2010) Research on the influence of machining introduced sub-surface defects and residue stress upon the mechanical properties of single crystal copper. *Sci China Technol Sci* 53(12):3161–3167
- Joshua AA, Chris DL, Alex T (2008) General purpose molecular dynamics simulations fully implemented on graphics processing units. *J Comput Phys* 227:5342–5359
- Harada T, Koshizuka S, Kawaguchi Y (2007) Real-time fluid simulation coupled with cloth. Theory and practice of, computer graphics, pp 13–20
- Harada T (2007) Real-time rigid body simulation on GPUs. GPU gems 3. Addison-Wesley, New York
- Shigeto Y, Sakai M (2011) Parallel computing of discrete element method on multi-core processors. *Particuology* 9(4):398–405
- Charles AR, Benjamin JG, Johannes GK (2010) Large-scale powder mixer simulations using massively parallel GPU architectures. *Chem Eng Sci* 65(24):6435–6442
- Nishiura D, Sakaguchi H (2011) Parallel-vector algorithms for particle simulations on shared-memory multiprocessors. *J Comput Phys* 230(5–1):1923–1938
- Zhu XS, Cheng L, Lu L, Teng B (2011) Implementation of the moving particle semi-implicit method on GPU. *Sci China Phys Mech Astron* 54(3):523–532
- Hori C, Gotoh H, Ikari H, Khayyer A (2011) GPU-acceleration for moving particle semi-implicit method. *Comput Fluids* 51(1):174–183
- Khayyer A, Gotoh H (2013) Enhancement of performance and stability of MPS mesh-free particle method for multiphase flows characterized by high density ratios. *J Comput Phys* 242(1):211–233
- Harada T, Koshizuka S, Kawaguchi Y (2007) Smoothed particle hydrodynamics on GPUs. In: Proceedings of computer graphics international, pp 63–70
- Harada T, Koshizuka S, Kawaguchi Y (2007) Slided data structure for particle-based simulations on GPUs. In: Proceedings of the 5th international conference on computer graphics and interactive techniques in Australia and Southeast Asia (GRAPHITE '07), Perth, Western Australia, pp 55–62

27. Alejandro JCC, Jose MD, Anxo B, Moncho GG, Benedict DR (2011) GPUs, a new tool of acceleration in CFD: efficiency and reliability on smoothed particle hydrodynamics methods. *PLoS one Public Libr Sci* 6(6):e20685
28. Daniel VB, Jose MD, Benedict DR, Alejandro JCC (2013) Towards accelerating smoothed particle hydrodynamics simulations for free-surface flows on multi-GPU clusters. *J Parallel Distrib Comput* 73(11):1483–1493
29. Jose MD, Alejandro JCC, Moncho GG (2013) Optimization strategies for CPU and GPU implementations of a smoothed particle hydrodynamics method. *Comput Phys Commun* 184(3):617–627
30. Ericson C (2004) Real-time collision detection (The Morgan Kaufmann Series in Interactive 3-D Technology). CRC Press, Boca Raton
31. Heinz TH, Hunenberger PH (2004) A fast pairlist-construction algorithm for molecular simulations under periodic boundary conditions. *J Comput Chem* 25(12):1474–1486
32. Mattson W, Rice BM (1999) Near-neighbor calculations using a modified cell-linked list method. *Comput Phys Commun* 119(2–3):135–148
33. Gonnet P (2007) A simple algorithm to accelerate the computation of non-bonded interactions in cell-based molecular dynamics simulations. *J Comput Chem* 28(2):570–573
34. Yao Z, Wang JS, Cheng M (2004) Improved $O(n)$ neighbor list method using domain decomposition and data sorting. *High Perform Comput Eng Syst (HPCES)* 161(1–2):27–35
35. Satish N, Harris M, Garland M, (2009) Designing efficient sorting algorithms for manycore GPUs. In: *Proceedings of the 2009 IEEE international symposium on parallel & distributed processing*, pp 1–10
36. Ha L, Kruger J, Silva TC (2009) Fast four-way parallel radix sorting on GPUs. *Comput Graph Forum* 28(8):2368–2378
37. Merrill GD, Grimshaw SA (2010) Revisiting sorting for GPGPU stream architectures. In: *Proceedings of the 19th international conference on parallel architectures and compilation techniques (PACT '10)*, Edmonton, Canada, pp 545–546
38. Hou Q, Zhou K, Guo B (2008) BSGP: bulk-synchronous GPU programming. In: *ACM transactions on graphics (TOG), proceedings of ACM SIGGRAPH 2008 TOG Homepage*, 27(3):Article No.19
39. Le GS (2007) Broad-phase collision detection with CUDA: GPU gems 3. Addison-Wesley, Boston
40. Tamai T, Koshizuka S (2014) Least squares moving particle semi-implicit method. *Comput Part Mech* 1(3):277–305
41. Dilts GA (1999) Moving-least-squares-particle hydrodynamics: I. Consistency and stability. *Int J Numer Methods Eng* 44:1115–1155
42. Liu WK, Jun S, Zhang YF (2005) Reproducing kernel particle methods. *Int J Numer Methods Fluids* 20:1081–1106
43. Ahuja RK, Magnanti TL, Orlin JB (1993) Network flows: theory, algorithms and applications. Prentice Hall, Englewood Cliffs
44. Ahuja RK, Mehlhorn K, Orlin JB, Tarjan RE (1990) Faster algorithms for the shortest path problem. *J Assoc Comput Mach* 37:213–223
45. Cherkassky BV, Goldberg AV, Radzik T (1993) Shortest paths algorithms: theory and experimental evaluation. Technical Report 93(1480), Computer Science Department, Stanford University
46. Fredman ML, Tarjan RE (1987) Fibonacci heaps and their uses in improved network optimization algorithms. *J ACM* 34:596–615
47. Gallo G, Pallottino S (1988) Shortest paths algorithms. *Ann Oper Res* 13:3–79
48. Hung MH, Divoky JJ (1988) A computational study of efficient shortest path algorithms. *Comput Oper Res* 15:567–576
49. Mondou JF, Crainic TG, Nguyen S (1991) Shortest path algorithms: a computational study with the C programming language. *Comput Oper Res* 18:767–786
50. Pallottino S (1984) Shortest-path methods: complexity, interrelations and new propositions. *Networks* 14:257–267
51. Zhan FB (1997) Three fastest shortest path algorithms on real road networks: data structures and procedures. *J Geogr Inf Decis Anal* 1(1):69–82
52. Zhan FB, Noon CE (1996) Shortest path algorithms: an evaluation using real road networks. *Transp Sci* 32(1):65–73