Data Management and Volcano Plume Simulation with Parallel SPH Method and Dynamic Halo Domains

Zhixuan Cao[1], Abani Patra[1], and Matthew Jones[2]

[1] Department of MAE, University at Buffalo, Buffalo, New York, U.S.A.
[2] Center for Computational Research, University at Buffalo, Buffalo, New York, U.S.A.

Abstract
This paper presents data management and strategies for implementing smoothed particle hydrodynamics (SPH) method to simulate volcano plumes. These simulations require a careful definition of the domain of interest and multi-phase material involved in the flow, both of which change over time and involve transport over vast distances in a short time. Computational strategies are developed to overcome these challenges by building mechanisms for efficient creation and deletion of particles for simulation, parallel processing (using the message passing interface (MPI)) and a dynamically defined halo domain (a domain that "optimally" captures all the material involved in the flow). A background grid is adopted to reduce neighbor search costs and to decompose the domain. A Space Filing Curve (SFC) based ordering is used to assign unique identifiers to background grid entities and particles. Time-dependent SFC based indices are assigned to particles to guarantee uniqueness of the identifier. Both particles and background grids are managed by hash tables which can ensure quick and flexible access. An SFC based three dimensional (3D) domain decomposition and a dynamic load balancing strategy are implemented to ensure good load balance. Several strategies are developed to improve performance: dynamic halo domains, calibrated particle weight and optimized work load check intervals. Numerical tests show that our code has good scalability and performance. The strategies described in this paper can be further applied to many other implementations of mesh-free methods, especially those implementations that require flexibility in adding and deleting of particles.

Keywords: SPH, MPI, data management, domain decomposition, domain adjusting, volcano plume

1 Introduction

Smoothed particle hydrodynamics (SPH) is a mesh-free scheme, initially developed for astrophysical applications and later extended to computational fluid mechanics. SPH has several features that match well with demands in geophysical flow simulation. It is easier in SPH to include multiple phases and handle complicated geometry and free boundaries. The first implementation of SPH in volcanology was presented by Bursik [1] with more developed recently

(eg.[5]). The development of a volcano plume (see Fig. 5) is essentially a multiphase turbulent ejection mixing process accompanied by microphysics phenomena like phase change of water, aggregation, reaction, etc. Accurate prediction of such complicated phenomena in a given time window requires resolution (very high particle counts) that can not be accomplished without parallel computing. Imposing some types of boundary conditions (such as wind field, eruption boundary condition at the vent) requires dynamically adding and removing particles during simulation. This requires an efficient and flexible data management scheme.

Most implementations of the parallel SPH method presented to date are limited to standard SPH and benchmark problems like dam breaks, or relatively simple scenarios like breaking-waves, flooding, etc. Work on more complicated implementations is relatively rare. Among existing CPU parallel SPH schemes, most of them focus on neighbor searching algorithms and dynamic load balancing. (eg. [4, 2]). Less attention has been paid to developing more flexible data management schemes. Fortunately, efficient and flexible data management strategies for parallel computing have been successfully implemented in mesh based methods (eg. [6] for adaptive hp finite element method (FEM), and [9] for finite volume method (FVM)). Motivated by these techniques developed for mesh based methods, we present a complete framework for parallelizing SPH program with the MPI standard allowing flexible and efficient data access.

Any implementation of SPH requires efficient searching and updating of neighbors during the simulation. Of the many possible choices we adopt the popular background grid method [7], which is also used for domain decomposition. We refer to the elements of the background grid as buckets. As for the actual storage of data representing the physical quantities associated to each particle, different strategies have been adopted in existing implementations of SPH. In DualSPHysics[2], the physical quantities of each particle (position, velocity, density...) are stored in arrays, and the particles (and the arrays with particle data) are reordered following the order of the cells. It is not flexible enough to add, delete and especially access particles for such fixed-size arrays. Ferrari[4] adopted linked lists using pointers so that particles can be deleted or added during the simulation. Storage problems caused by fixed-size arrays are thereby also eliminated. We adopt hash tables to store pointers to objects of particles and buckets, which give us not only flexibility of deleting and adding of elements, but also quicker access compared with linked lists. A SFC-based index is adopted to give each particle and background bucket an unique identifier – a strategy known to preserve some locality at minimal cost. The SFC based numbering strategy is further extended to include time step information so that particles added at the same position but different time will have different identifiers. We adopt an easy-programming scheme based on SFC [10] to decompose the domain.

To the best of the author's knowledge, no current implementation of SPH has the feature of adjusting the computational domain based on simulation needs. For volcano plume simulation, this feature will greatly reduce computational cost by avoiding computing of uninfluenced fluids. This feature is accomplished by adding a scan function to monitor the outermost layer of the domain and turn ghost particles to real particles at the proper time thus effectively defining a halo domain with an almost optimally defined size.

2 Data Structure and Load Balance

In SPH, the domain is discretized by particles and the position of each particle is updated at every time step. The physical laws written in the form of partial differential equations need to be transformed into the Lagrangian particle formalism of SPH. Using a kernel function that provides the weighted estimation of the physical properties in the neighborhood of a

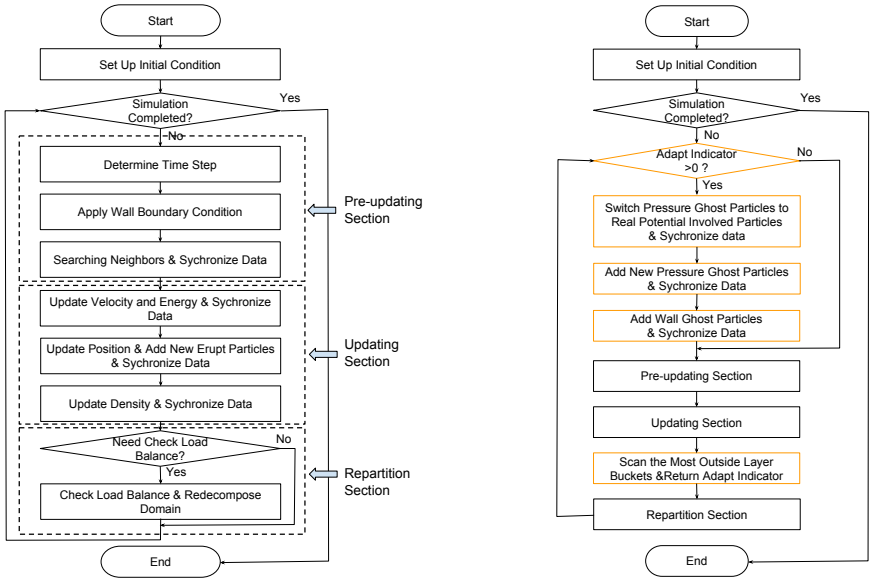

Figure 1: Workflow for SPH code. Figure to the left is the basic workflow. The right figure is the workflow that enables dynamic halo domain.

discrete point (particle), the integral equations are evaluated as sums over neighbor particles. Only particles located within the support of the kernel function will interact. Thus, physical properties associated with the particle are updated based on its neighbors. A neighbor search therefore needs to be carried out before updating physical properties. A basic workflow of our SPH code is shown in Fig. 1. We use buckets, which contain all particles associated with a sub-domain (see Fig. 2) and keep them stationary during the entire simulation, to reduce searching cost (since the search can now be restricted to only neighboring domains). Domain is decomposed based on a SFC going through centroids of all buckets, as shown in Fig. 2.

2.1 SFC based indexing

Our data structure starts from assigning each particle and bucket an identifier, we refer to it as a key, which should be unique throughout the simulation. The key for a bucket is determined by the centroid coordinate of the bucket while the key for a particle is determined by adding coordinates and time step of the particle. The map from coordinates to key is based on SFC [11] which maps a n-dimensional domain to a one dimensional sequence. The standard procedure for obtaining the SFC is:

- Scale all coordinates in computational domain ($\mathbf{X}'$) into an n dimensional unit hypercube $[0,1]^n$: $\mathbf{X}' \rightarrow \mathbf{X}$, where $\mathbf{X}$ represents coordinates in the hypercube.
- Compute $k_r = h_n(\mathbf{X})$. Where h_n is the map $h_n : [0,1]^n \rightarrow [0,1]$.
- Convert k_r to integer k by multiplying k_r with a large integer and removing decimal part.
- Sort all keys to form a SFC sequence. The SFC represents a curve passing through all particles (or centroids of buckets).

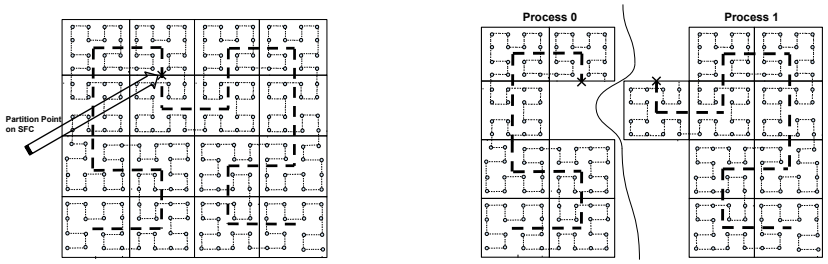

Figure 2: The left figure shows SFCs passing all particles and buckets. The right figure shows an example of a domain decomposition based on the SFC of buckets.

Details for constructing the map h_n can be found in [8]. These keys denote a simple addressing and ordering scheme for the data, i.e., a global index space for all the objects.
In some situations, new particles need to be added during the simulation. For example, new particles need to be added at the bottom of the eject vent (see Fig. 3). To assign distinct identifiers for particles added at the "same position" but at different time steps, we extend the SFC-based key to a time-dependent SFC based key by including date of birth of particles into their keys. The time-dependent SFC based key can be denoted as: $[k, t]$, where t is the time step. The map h_n will become:

$$h_n : [0,1]^n \times \mathbf{T} \rightarrow [0,1] \times \mathbf{N} \tag{1}$$

Where $\mathbf{T} \subset [0, \infty)$ is the time dimension, and $\mathbf{N} = \{0, 1, 2, 3...\}$. To guarantee locality, sorting of particle keys is based on k, that is to say, particles with smaller k always come before particles with larger k. For particles that have the same k, ordering will then depend on t. Figure 2 shows SFC ordering of buckets and particles in buckets. Several features of this indexing scheme are suitable for SPH:

- Guaranteed uniqueness of keys.
- The key of each object is generated purely based on its own coordinates. When we add new objects on different processes, the key of each object can be generated fast and independently.
- Objects that are located closely in the Euclidian space will also be close to each other in the one dimensional SFC key space in a mean sense. Since SPH particles only interact with their neighbors, geometric locality can be exploited for efficient access of data.
- This type of key effectively generates a global address space. Globalism of key and conservation of spatial locality make it easy to partition the sorted key sequence and obtain a decomposition of the problem.

2.2 Data management strategies

The most basic data structure of SPH are particles and buckets. Both are defined as C++ classes. Information that is contained in particle objects can be categorized into six categories: identifier (the key), affiliate(rank of the process that the particle belongs to), primitive variables (variables in governing equations, e.g. density, velocity), secondary variables (physical properties that can be computed from primitive variables, they are stored to avoid repeated

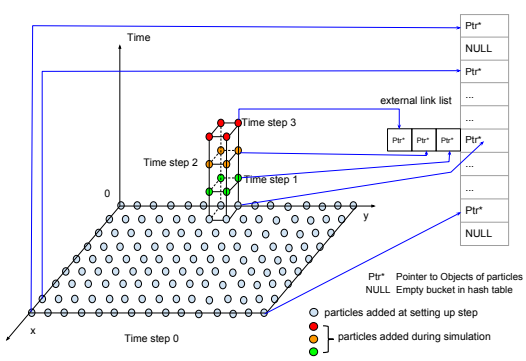

Figure 3: Non-uniform distribution of particles in the $[0,1]^n \times \mathbf{T}$ space due to adding of new particles at a small area of the whole domain.

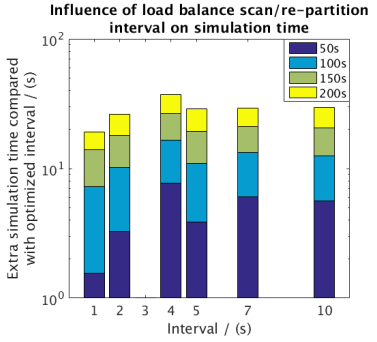

Figure 4: The influence of different load balance check intervals on simulation time. The bar values are the extra simulation time in log scale. The optimized interval is 3s.

computations, e.g., pressure and temperature), flags (indicators, such as indicator for ghost particle or real particle) and neighbor information (it is a vector of particle keys in our application). Similarly, information that is contained in a bucket object can also be categorized into different categories: identifier, affiliate, dimension information (maximum and minimum coordinates, boundary information), flags (indicators, such as guest indicator, which is used to indicate whether the bucket belongs to another adjacent subdomain or not), neighbor information (keys of 27 neighbor buckets including its own key) and owned particles (it is a vector of particle keys in our application). Objects are then accessed through hash tables.

One of the fundamental data structures that allows flexible data access is a hash table. Based on the keys of data, the address-calculator(hashing) function determines in which slot the data should be stored:

$$Index = hash(key) \tag{2}$$

Data in a hash table can be accessed with time complexity of O(1) when there is no hash collision. One way to handle hash collisions is to use an additional sorted vector attached to the hash table. When several keys hash to the same index, a vector will be created. The vector is sorted based on keys so that a binary search (with an average time complexity of O(log n)) can be used to find the correct position for adding, deleting or retrieving of data. Another option is using an additional linked list which is more flexible in memory allocation but slower in searching (O(n)). In addition, pointer based memory accessing of linked lists will greatly slow down the efficiency of data access for long linked lists. Choosing the proper way to handle hash conflicts greatly depends on the problem itself. For volcano plume simulation, new particles are added at the bottom of the eruption vent (see Fig. 3), making the linked list a better choice. Pointers to these new particles will be stored in external linked lists as shown in Fig. 3. For time-independent keys, the hash function can be a simple function like:

$$Index = \frac{Key - Min\,Key}{Max\,Key - Min\,Key} \times Hash\,Table\,Size \tag{3}$$

One natural way to hash time-dependent keys $[k,t]$ is to convert the two elements in the key into one number taking k as the higher digit and t as the lower digit of the large number. However, for volcano plume simulation, only ghost particles for eruption boundary condition will be successively added at the same place: the bottom of the vent. That is to say, particles

Table 1: Computational cost per particle for different steps

Step	Cost (ms)	Real	wall	eruption	pressure
neighbor search	0.41	Yes	No	No	No
update momentum and energy	0.70	Yes	No	No	No
update density	0.42	Yes	No	No	No
update position	0.01	Yes	No	Yes	No
velocity filtering	0.43	Yes	No	No	No
apply wall boundary condition	0.75	No	Yes	No	No
summation (ms)	-	1.97	0.75	0.01	0.00

are distributed non-uniformly in the $[0,1]^n \times \mathbf{T}$ space as shown in Fig. 3. To avoid a non-uniform sparse hash table and conserve spatial locality, we only plug the first number, k, of the key, $[k,t]$, into the hashing function, Eq. (3). All particles with the same birth location will hash to the same index and be handled by external linked lists.

2.3 Domain decomposition and load balancing strategy

Particles in our problem can be categorized into four types: real particles, wall ghost particles, pressure ghost particles and eruption ghost particles. Ghost particles are for imposition of corresponding boundary conditions (See Fig. 5). As different types of particles involve different amount of computational work, shown by table 2.3, we assign different work load weights for different types of particles based on profiling data. Instead of simply using number of contained particles as work load for buckets, work load of each bucket is determined by summing up work load weights of all particles within the bucket. The SFC of buckets now becomes a weighted sequence. Domain decomposition is conducted by cutting the weighted SFC of buckets into pieces. Each piece of the SFC should then have a balanced total work load.

Figure 2 shows how the domain is decomposed based on partition of the SFC of buckets. The particles are automatically split into several groups along with buckets that contain them. As the SFC of buckets is a curve in three dimensional (3D) space, partition of this curve will automatically lead to 3D domain decomposition. Movement of particles, addition of new particles, and adjustments of domain will lead to significant load imbalance between processes. To restore load balance, computational load is monitored at a given interval which is optimized based on numerical experiments. Repartitioning is carried out when load imbalance is larger than a given tolerance.

As some of the neighbor particles reside in other partitions, a set of "guest" particles and buckets are used to synchronize data across partitions. To minimize communications, data is synchronized only where needed, using non-blocking MPI communications.

3 Dynamic Halo Domain

For plume simulation and similar phenomena, where some fluid ejects into a stationary fluid and gets mixed, the stationary fluid must be represented. A lot of CPU time will be spent on computing associated with these stationary particles. If simulation of stationary particles can be avoided, the computational cost will be greatly reduced. We propose a simple strategy to add such a feature to our code with low computational cost. We add a scan function to monitor the outermost layer of the domain. When the ejected fluid reaches the boundary of the current domain, pressure ghost particles (light blue particles in the right figure of Fig. 5)

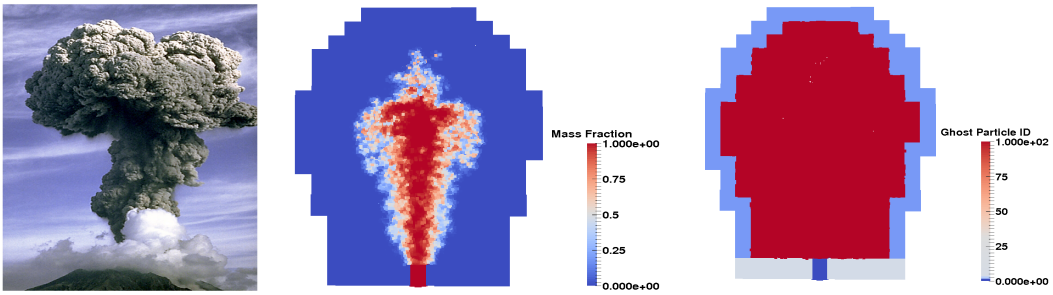

Figure 5: The left figure shows a plume photo of Sakurajima Volcano eruption, the middle figure shows a typical simulation results with the same input parameters as "Run P1" in reference [12]. The right figure is a cross-section view of the computational domain, it illustrates how particles are deployed on different boundaries: red are real particles, dark blue are eruption ghost particles, grey are wall ghost particles, the light blue are pressure ghost particles.

will be turned to real particles (red particles in the right figure of Fig. 5) and then add a new layer of pressure ghost particles for pressure boundary conditions. Wall ghost particles (grey particles in the right figure of Fig. 5) will also be added when necessary. The original workflow is modified to enable such a feature. The modified workflow is shown on the right in Fig. 1).
An involvement flag is added to particle class to distinguish different states of involvement. Particles are categorized into three groups based on the value of the involvement flag: involved, potentially involved, and not involved. The involved particles are particles that have already been affected by the mixing. The potentially involved particles are particles that have not been involved in mixing but are adjacent to involved particles and will thus be involved in the near future. All communication and computation associated with uninvolved particles can be ignored. That is to say, only potential involved and involved particles need to be simulated. As simulation progresses, the ejected fluid will reach a larger area and more and more particles will be influenced. When originally stationary air is influenced by erupted material, the mass fraction of the erupted material will increase from zero to a positive value. So we can determine whether a particle should turn to involved status based on whether the mass fraction of that particle is larger than a given threshold (10^{-5} in our simulation) or not. Other physical properties, like velocity, can also serve as alternative "switch criteria."
A similar scheme is also deployed for buckets resulting in a categorization of buckets. The additional complexity is that potentially involved ghost particles must also be taken into accounted.

4 Numerical Test

We use the same governing equations and input parameters as the plume simulation presented by Suzuki [12]. A typical simulation result of volcano plume by our code is shown in Fig. 5.

4.1 Scalability test

Experiments have been carried out on the computational cluster of Center for Computational Research (CCR) at Buffalo. The compute servers have 12 cores (two Xeon E5645 processors per server) running at 2.40GHz clock rate with 4GB memory per core on a Q-Logic Infiniband network. Main memory and level 3 cache are shared on each node. The simulation domain is

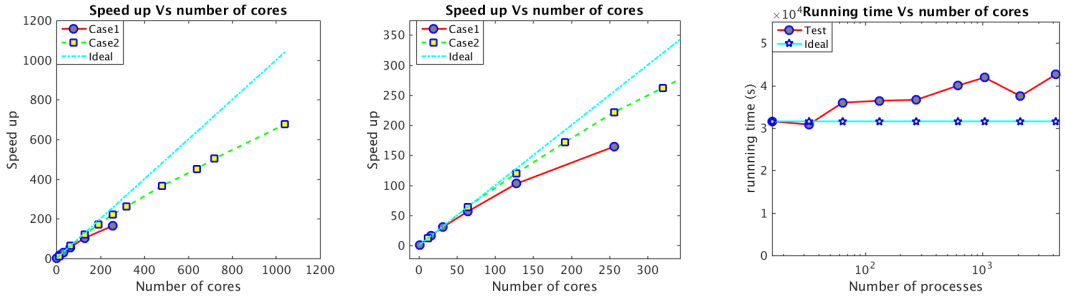

Figure 6: The left figure shows strong scalability tests result. middle figure is the zoomed view of first one. It is obviously shown that strong scalability is better when the problem size is larger. The right figure is weak scalability test results

Table 2: Simulation time for the same particle weight and different particle weights at different physical times.

Physical time	10 s	20s	30 s	40 s
Same weight	1141.7	4119.4	10371.0	12453.7
Different weights	1108.2	4057.0	10281.5	12166.3

a box with initial dimension $[-4.8km, 4.8km] \times [-4.8km, 4.8km] \times [0km, 6km]$ for case 1 and $[-10km, 10km] \times [-10km, 10km] \times [0km, 5km]$ for case 2. The smoothing length (we set initial intervals between particles equal to smoothing length) equals to 200m and 100m respectively for test case 1 and test case 2. So the computational work load of test case 2 is larger than that of the test case 1. The simulations run for 20s physical time. Parallel speed up is shown in Fig. 6. Test case 2 shows better speed up than test case 1 which implies that strong scalability can be improved by increasing total amount of work load. The weak scalability test is conducted with the same initial domain as test case 1 and various smoothing length. Each simulation runs for 400 time steps. The average number of real particles for each process keeps constant at 25900, so the average work load for each processor is approximately constant. Right side figure in Fig 6 shows the timing of a weak scalability test.

4.2 Effect of workload check interval and calibrated particle weight

In section 2.3, we proposed to use different particle weights for different types of particles when estimate the workload of buckets. The effect of using different particle weight is demonstrated in table 4.2. The simulation time is reduced by using different particle weights. However, the amount of time that is reduced is not as significant as we expected. One possible explanation is that real particles are the major population, so the load imbalance caused by assigning improper weight values to ghost particles is small.

The best load balance check interval is calibrated based on a series of simulations with different load balance scan intervals. From 0 - 50 seconds physical time, the interval of 1 second shows a better load balance than interval of 2 seconds. However, for the simulation of 50 - 100 seconds, interval of 2 seconds is better. This implies that loss of load balance accumulates faster and requires a more frequent re-decomposition of domain at the early stages of simulation. This is consistent with the plume development process: the domain grows quickly at the beginning as

Table 3: Computational work load of extra steps for domain adjusting. SWCH represents step that switch pressure ghost particle to real particle, ADPP is short for adding new pressure ghost particles, ADWP represents adding wall ghost particles, SCN is short for scanning the outmost layer of the domain. Momentum and energy update (UPME) and position update (UPP) also included for comparison.

Functions	Total time (s)	Called times
UPME	2954.8	201
UPP	38.55	201
ADPP	21.51	3
ADWP	8.88	3
SWCH	0.08	2
SCN	7.72	201

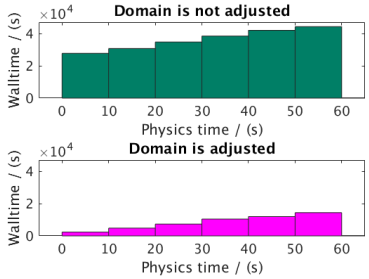

Figure 7: The figure on the top shows execution time without domain adjusting, the figure on the bottom shows execution time with domain adjusting. Different bins represent execution time up to specific physical time indicated by horizontal axis.

erupted material ejects into the environment and spreads. Afterwards, the spreading speed of the front edge slows down due to viscosity and momentum exchange leading to slowing down of domain growth. We need to emphasize that, these optimized parameters, including weights of particles and load balance checking interval, are case sensitive and need to be re-calibrated for different applications and hardware architecture.

4.3 Effect of dynamic halo domains

For the test problem in this paper, the volcano plume finally reaches to a region of $[-10km\ \ 10km] \times [-10km\ \ 10km] \times [0km\ \ 20km]$ after around 300 seconds of eruption. When numerical simulation goes up to 90 seconds, the plume is still within a region of $[-3km\ \ 3km] \times [-3km\ \ 3km] \times [0km\ \ 6km]$. This implies that adjusting of domain can avoid computing large number of uninfluenced particles, especially for the early stages of simulation. On the other hand, additional functions (see Fig. 1) for domain adjusting will add extra computational cost. Table 3 shows the extra computational cost corresponding to these functions. Profiling data for momentum and energy update (UPME) and position update (UPP) is also included in the table for the purpose of comparison. As we can see from the table, the cost of SCN is even smaller than UPP function, which is the cheapest function (as shown in table 2.3) in the regular workflow (see Fig. 1). The other three functions, ADPP, ADWP and SWCH, are only called a few times during the simulation, and as a consequence, the extra computational cost due to them are negligible. To summarize, the additional cost caused by the domain adjusting functions is ignorable. Figure 7 shows that simulation time of the test problem is greatly reduced when we adopt the domain adjusting strategy in our code.

5 Conclusion

We developed data management strategies for a MPI-parallel implementation of the SPH method to simulate volcano plumes. Neighbor searching and domain decomposition is based on the background grid. A SFC based index scheme is adopted to identify buckets and time-dependent SFC keys are used as identifiers for particles. Hash tables with external linked lists

are adopted for accessing particles and buckets data. Based on calibrated particle weights, a dynamic load balance strategy is developed by checking load balance and re-decomposing the domain at an optimized interval. The performance of the code was further improved by several factors by adjusting the computational domain during simulation. Overall we obtain good scalability and performance.
The flexibility of our data access methodology enables implementing mesh-free methods for solving more complicated problems and using more advanced techniques, such as dynamic particle splitting techniques[13, 3], which will give higher resolution at the area of interest by splitting one large particle to several smaller ones. The data structure, particle and bucket indexing strategies, domain decomposition, dynamic load balancing method and domain adjusting strategies in this paper can be adopted to other mesh-free methods (not just SPH).

References

[1] M Bursik, B Martınez-Hackert, H Delgado, and A Gonzalez-Huesca. A smoothed-particle hydrodynamic automaton of landform degradation by overland flow. *Geomorphology*, 53(1):25–44, 2003.

[2] AJC Crespo, JM Domínguez, BD Rogers, M Gómez-Gesteira, S Longshaw, R Canelas, R Vacondio, A Barreiro, and O García-Feal. Dualsphysics: Open-source parallel cfd solver based on smoothed particle hydrodynamics (sph). *Computer Physics Communications*, 187:204–216, 2015.

[3] J Feldman and J Bonet. Dynamic refinement and boundary contact forces in sph with applications in fluid flow problems. *International Journal for Numerical Methods in Engineering*, 72(3):295–324, 2007.

[4] Angela Ferrari, Michael Dumbser, Eleuterio F Toro, and Aronne Armanini. A new 3d parallel sph scheme for free surface flows. *Computers & Fluids*, 38(6):1203–1217, 2009.

[5] Bouchra Haddad, David Palacios, Manuel Pastor, and José Juan Zamorano. Smoothed particle hydrodynamic modeling of volcanic debris flows: Application to huiloac gorge lahars (popocatépetl volcano, mexico). *Journal of Volcanology and Geothermal Research*, 324:73–87, 2016.

[6] Andras Laszloffy, Jingping Long, and Abani K Patra. Simple data management, scheduling and solution strategies for managing the irregularities in parallel adaptive hp finite element simulations. *Parallel Computing*, 26(13):1765–1788, 2000.

[7] Joseph J Monaghan and John C Lattanzio. A refined particle method for astrophysical problems. *Astronomy and astrophysics*, 149:135–143, 1985.

[8] Abani Patra and J Tinsley Oden. Problem decomposition for adaptive hp finite element methods. *Computing Systems in Engineering*, 6(2):97–109, 1995.

[9] Abani K Patra, AC Bauer, CC Nichita, E Bruce Pitman, MF Sheridan, M Bursik, B Rupp, A Webber, AJ Stinton, LM Namikawa, et al. Parallel adaptive numerical simulation of dry avalanches over natural terrain. *Journal of Volcanology and Geothermal Research*, 139(1):1–21, 2005.

[10] AK Patra and DW Kim. Efficient mesh partitioning for adaptive hp finite element meshes. In *In International Conference on Domain Decomposition Methods*. Citeseer, 1999.

[11] Hans Sagan. *Space-filling curves*. Springer Science & Business Media, 2012.

[12] Yujiro J Suzuki, Takehiro Koyaguchi, Masaki Ogawa, and Izumi Hachisu. A numerical study of turbulent mixing in eruption clouds using a three-dimensional fluid dynamics model. *Journal of Geophysical Research: Solid Earth*, 110(B8), 2005.

[13] R Vacondio, BD Rogers, and PK Stansby. Accurate particle splitting for smoothed particle hydrodynamics in shallow water with shock capturing. *International Journal for Numerical Methods in Fluids*, 69(8):1377–1410, 2012.