Dynamic Load Balancing for SPH

Pushkar Godbole
Department of Aerospace Engineering
Indian Institute of Technology Bombay
Powai, Mumbai 400076
Email: pushkar.godbole91@gmail.com

Kunal Puri
Department of Aerospace Engineering
Indian Institute of Technology Bombay
Powai, Mumbai 400076
Email: kunal.r.puri@gmail.com

Prabhu Ramachandran
Department of Aerospace Engineering
Indian Institute of Technology Bombay
Powai, Mumbai 400076
Email: prabhu.ramachandran@gmail.com

***Abstract*—We evaluate the performance of different load balancing algorithms used in the open source SPH framework PySPH. We focus on algorithms that are used to generate particle decompositions that adapt to the dynamic nature of the simulation as opposed to spatial domain decomposition methods. A synthetic test case for particles randomly distributed in a box and the (2D) dam break problem with an obstacle are used as test cases to measure the efficiency of the algorithms and quality of the partitions. The relatively simple geometric algorithms are found to perform well in this context with the Recursive Inertial Bisection (RIB) method producing the best partitions for the problems considered.**

I. Introduction

Smoothed Particle Hydrodynamics (SPH) is a meshless method employing the use of points/particles as discretization entities for the numerical simulation of continuum mechanics problems. The governing PDEs are transformed into ordinary differential equations for particle field variables, constraining individual particles to interact with near neighbors. The subsequent motion of particles with the local material velocity renders the method Lagrangian. As a numerical tool, SPH is a competitive option for large deformation problems in arbitrary domains for problems in astrophysics, solid-mechanics and incompressible fluid flow [1]–[3]. The principle application area of SPH remains free surface flows in which an incompressible fluid moves under the influence of gravity in an open vessel [4]–[6]. In comparison to mesh based methods, SPH is characterized by requiring a large number of entities (particles) for an accurate description of the flow and more computations per entity than their mesh based counterparts. Furthermore, the explicit nature of most SPH implementations imply stringent time step criteria for stability which further increases the computational time required. In these circumstances, one must look for parallel paradigms for any realistic SPH implementation. The choice for parallelism is complicated by the vast array of available hardware ranging from clusters to hybrid computing environments that leverage the processing power of massively parallel graphics processing hardware (GPUs) [7]. In general, parallel implementations can be categorized as data or task parallel. Domain decomposition falls into the data parallel model wherein the particles are distributed across available processors in a distributed computing environment. The processor assignment can either be spatial according to the domain or adapt to the particle distribution. In this work, we focus on the class of decomposition algorithms that adaptively re-distribute particles in an arbitrary domain. Specifically, we discuss the class of geometric load balancing algorithms that rely on coordinate bisection, space filling curves or a k-means clustering approach. Although the discussion on load balancing is application agnostic, we use the 2D dam break problem in the presence of an obstacle as a concrete test case. The choice of this problem is justified considering that the problem adds a sufficient amount of complexity like multiple species (fluid, solid) and boundary conditions without obfuscating the parallel implementation with the details of the SPH algorithm. More generally, the discussion in this work is applicable to any dynamic simulation with short range particle interactions. We work within the framework of PySPH and use the Zoltan data management library [8], [9] for the bisection and space filling curve partitioners. The algorithms are assessed on the execution times and quality of partitions in terms of the number of particles exchanged per iteration.

The outline of the paper is as follows. In Sec. II, we describe the parallel algorithm for distributed memory machines we adopt for PySPH. We briefly describe some of the load balancing algorithms applicable to SPH in Sec. III before going on to discuss the results for the algorithms implemented for the box test and a dam break problem with an obstacle in Sec IV. Finally, we summarize and conclude this work in Sec. V.

II. Smooth Particle Hydrodynamics

We outline a typical sequence of SPH operations for a problem and the parallel model for the implementation adopted in PySPH. The basic assumption for the model is a distributed memory machine equipped with the message passing library (MPI).

A. Serial SPH

SPH can be introduced via the theory of kernel interpolation whereby a sufficiently smooth function f can be approximated at an arbitrary point $\boldsymbol{x}$ in a domain Ω by summing contributions from near-neighbors

$$f(\boldsymbol{x}) \approx \sum_{b \in \mathcal{N}(\boldsymbol{x})} f_b W(|\boldsymbol{x} - \boldsymbol{x}_b|, h) \Delta_b \tag{1}$$

The interpretation is that the function values are weighted by a kernel function W, and Δ_b is the nodal volume associated with an interpolation point. The kernel is compactly supported

which implies a summation over nodes only in the neighborhood of the evaluation point $\boldsymbol{x}$ (denoted as $\mathcal{N}(\boldsymbol{x})$) as shown in Fig. 1 Gradients can be constructed in similar vein to result in

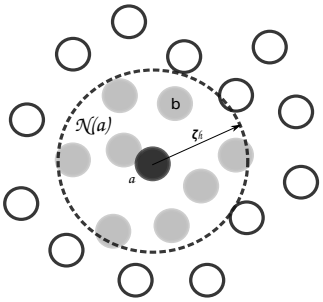

Fig. 1. The set of neighbors ($\mathcal{N}_a$) for a given particle is determined by it's interaction radius.

evolution equations in the form of short-range forces between particle pairs. We refer the interested reader to [10], [11]. An example of (1) is to approximate the density at a point as

$$\rho_a = \sum_{b \in \mathcal{N}(a)} m_b W(\boldsymbol{x}_a, \boldsymbol{x}_b, h_a) \tag{2}$$

This is a common test to verify the correctness of an SPH implementation. In this equation, the function f is the particle density (ρ) and the nodal volumes are approximated as the ratio of particle mass to density ($\Delta_b = m_b/\rho_b$).
For free surface problems, we solve the incompressible Navier-Stokes equations written in Lagrangian form

$$\frac{d\rho}{dt} = -\rho\,(\nabla \cdot \boldsymbol{v}) \tag{3a}$$

$$\frac{d\boldsymbol{v}}{dt} = \boldsymbol{g} - \frac{1}{\rho}\,(\nabla p) + \mathcal{F}_{\text{visc}} \tag{3b}$$

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{v} \tag{3c}$$

using the weakly compressible approach [4], [12]. The pressure is coupled to the density using the Tait equation of state

$$P = B\left(\left(\frac{\rho}{\rho_o}\right)^{\gamma} - 1\right) \tag{4}$$

The no penetration boundary condition can be imposed either using repulsive particles [4] or by using dynamic boundary particles [6]. What is essential is that the evolution of the system is defined solely through pairwise interactions. The equations are integrated using a second order predictor corrector or Runge-Kutta integrator. The sequence of operations to update this system involves a partial update to a half time step, computation of accelerations and subsequent integration to the full time step. We refer to this relatively benign set of operations as the *serial* part of the algorithm.

B. Parallel SPH

In our parallel SPH implementation, every processor is responsible for the evolution of a subset of the total number of particles. The particles assigned to a processor are called *local*, and to carry out the short-range force computations across processor boundaries, *remote* particles are shared as buffer or halo regions as shown in Fig. 2. We assume that

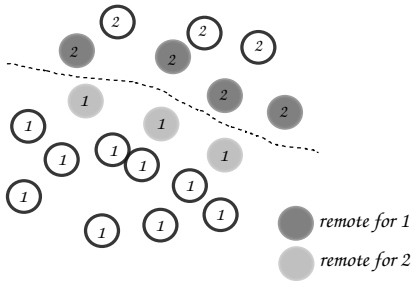

Fig. 2. Local and remote particles for an imaginary particle distribution. Particles assigned to a processor are *local* while those shared for neighbor requirements are *remote*

each particle requires the same processing time per time step, which is valid for the weakly compressible formulation. With this assumption, the number of local particles on a processor determines the balance of load while the number of remote particles exchanged provides a measure of the quality of a partition. Owing to the Lagrangian nature of SPH, the particles move over the computational domain and the remote particles must be re-computed at every time step to satisfy the neighbor requirements. Dynamic load balancing is needed to achieve a good partition to keep the communication overhead low. Once the ghost particles have been exchanged for a given decomposition, the serial algorithm can be applied on each processor to update the local particles.
The sequence of operations for the parallel SPH implementation is listed in Algorithm. 1. We assume some given decomposition as initial input. If the partition is not balanced (based on estimates available), we call the load balancer to redistribute the particles. Remote particles are then exchanged to satisfy neighbor requirements. Once the new decomposition is available, we perform the serial SPH calculations to determine the forces on the local particles. When the particle

Algorithm: Parallel Update
if *not balanced* **then**
  call LoadBalance
  call ComputeRemoteParticles
end
call SerialSPH
for *each local particle* **do**
  call FindNewPartition
  call UpdatePartition
end

Algorithm 1: Update sequence in our parallel SPH implementation

positions are updated, we determine the partition to which they must be assigned and update it. Counting the number of particles to be re-assigned gives us a measure to check if load

balancing is required in a subsequent time step. This parallel update algorithm is straightforward and hinges on an efficient dynamic load balancing technique. Some of these techniques are discussed next.

III. Load Balancing Algorithms for SPH

Load balancing algorithms strive to achieve a balanced workload among available processors while minimizing the communication overhead for the additional data required for the calculation. For SPH, the number of local particles assigned to a processor determines the workload and the exchange of remote particles can be used as a metric for the communication overhead. The general load balancing problem can be described as the assignment of "objects" to processors to meet the load balancing objectives. The objects can be interpreted quite generally as the particles themselves or cells representing an underlying indexing structure used for neighbor queries [13]. The input for the geometric load balancing algorithms are the physical coordinates of the objects while graph based partitioners require connectivity information between the objects.

Using the cell structure as the objects to be partitioned is preferred since the cells provide an agglomeration of different species in a given spatial region and can be conveniently interpreted as a mesh with fixed connectivity irrespective of the number of particles within a cell. To account for cells with variable number of particles, weights are assigned to reflect this skewed distribution of cells. In the following, we briefly describe the load balancing algorithms used in this work.

A. Geometric Partitioners

Geometric partitioners take as input, the coordinate positions of the objects to be partitioned. The algorithms are physically intuitive and are the natural choice for particle methods like SPH where either the particle coordinates or the centroids of the cell indexing structure can be used as inputs to the algorithm.

1) Recursive Coordinate Bisection (RCB): The RCB algorithm is the simplest of the geometric partitioners. It begins by bisecting the domain using a plane orthogonal to the coordinate axes. This generates two sub-regions with equal number of objects. The algorithm is then applied recursively to each sub-region to reach the desired number of partitions. Weights can be assigned to objects to achieve the load balancing criteria. RCB has been used by Fleissner and Eberhard [15] for coupled SPH BEM simulations. They propose to dynamically update the partitions generated by the algorithm to adapt to the changing particle distribution. The principle advantage of the RCB method is speed and simplicity. A disadvantage of the RCB method is that it works best for axis aligned domains and therefore can generate disconnected sub-domains with an increasing number of partitions.

2) Recursive Inertial Bisection (RIB): A variant of the RCB method that works well for non axis aligned domains is the recursive inertial bisection (RIB) method. In this method, the bisection plane is orthogonal to the principle inertial axis of the objects to be partitioned. The method produces visibly better partitions than RCB for arbitrary domains at the expense of slightly longer running times owing to the eigenvalue computations for the principle axes.

3) Hilbert Space Filling Curve (HSFC): Space Filling Curves (SFCs) have been used by Springel [16] (GADGET) to partition a hierarchical tree used for long range gravitational calculations and short range pairwise SPH forces. SFCs are functions that map a given point in a domain into the unit interval $[0, 1]$. The inverse functions map the unit interval to the spatial coordinates. The HSFC algorithm divides the unit interval into P intervals each containing objects associated to these intervals by their inverse coordinates. N bins are created to partition the interval and the weights in each bin are summed across all processors. The algorithm then sums the bins and places a cut when the desired weight for a partition is achieved. In this work, we use the Hilbert SFC to partition the objects although in principle, any space filling curve can be used.

4) K means clustering: Clustering approaches have been used in data mining and analysis to group or partition observations (objects) into clusters. In the k-means approach to agglomeration, the decision to associate an object with a cluster is an optimization process aimed at minimizing a cost function that depends on the distances of objects to cluster centroids in a given metric. Marzouk and Ghoniem [17] used a weighted k-means technique to obtain domain decompositions for particles in a hierarchical vortex method. The simplest metric for k-means clustering can be the standard Euclidean norm, however, this results in unbalanced workloads as the size of the clusters are not taken into account. A better partition can be achieved by using a norm that scales the distance inversely as the size of the cluster grows. The algorithm is iterative and is easy to parallelize. Our experiments show that it takes roughly $20 - 30$ iterations for the algorithm to converge. The partitions produced by k-means for regular domains are similar to those produced by recursive coordinate bisection (RCB).

B. Graph Partitioners

Graph partitioning techniques used for mesh based schemes re-interpret the mesh data structure as a graph, with the elements defining the nodes and the element connectivity as the edges between the nodes. The algorithm then proceeds to partition the graph by performing cuts along the edges. Since the connectivity information is implicit in the definition of a graph, the application of these algorithms to meshless methods is not clear. In SPH, the particles could be treated as the vertices in the graph while the nearest neighbors could define the edges emanating from a given vertex. While this works in principle, the resulting graph can have too many edges, leading to expensive running times for the algorithm. An alternative is to use the particle indexing structure as an implicit mesh where the cells define the nodes and the cell neighbors (27 in 3D and 9 in 2D) define the node connectivity [5]. The graph is now invariant to the number of neighbors for

a given particle (resolution in SPH). Edges can be weighted based on the number of particles in each cell to achieve a good load balance across processors. The cells however, must be unique across the computational domain which raises the communication overhead. Additional overhead is incurred for these methods in determining the remote particles required to complete the load balancing phase. The main advantage of graph partitioners is that they produce fast and high quality partitions.
For our experiments, we use the serial graph partitioner Metis [14] to obtain a distribution of particles which are then scattered to their respective processors. Other graph partitioning algorithms that arguably produce better results are the class of hyper-graph partitioners. The inclusion of these techniques in the discussion was beyond the scope of this work.

IV. Examples and Results

In this section we compare the particle distributions produced by the different load balancing algorithms for two test cases. The first test is a synthetic test in which particles are randomly distributed in a box. The second test is the evolution of a breaking dam in 2D as it encounters an obstacle in it's path. While the results presented are for two dimensions, the implementation is readily applicable in three dimensions. The 2D examples are chosen to facilitate an understanding of the distributions produced by the different algorithms.

A. The box test

The first example is a *box-test* in which particles are randomly distributed in the unit square. The initial particle to processor assignment is random, which generates a highly imbalanced distribution with which we check the performance of the different load balancing algorithms. Fig. 3 shows representative partitions generated using the RCB, RIB, HSFC and Metis partitioners for 2^{16} particles on 4 processors. Although

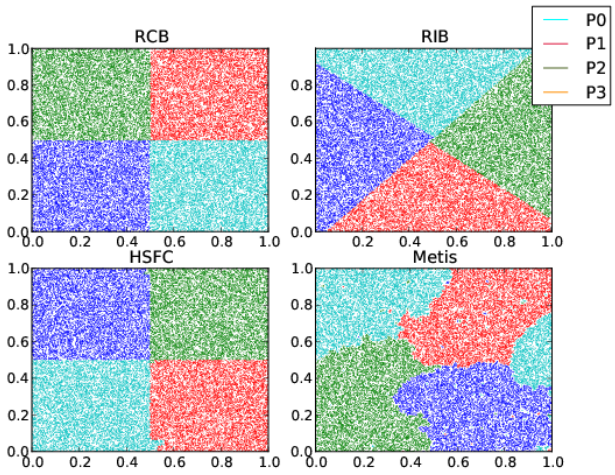

Fig. 3. Particle distribution for 2^{16} particles on 4 processors. Particles are colored according to processor assignment.

this is a very simple domain, the differences between the algorithms is evident. The RCB method produces sub-domains that are axis-aligned. The RIB method has performed cuts along the diagonal of the domain which results in the triangular sub-domains. The HSFC method produces a partition very similar to RCB but with a slight overlap of processor domains. Metis has produced a very irregular distribution with disconnected regions. We note that we used particle coordinates as the graph vertices and particle neighbors as the graph edges for the input to Metis. An alternative would be to use the cell indexing structure as an input mesh for Metis. Fig. 4 shows the partitions generated on 8 processors for the same problem. The

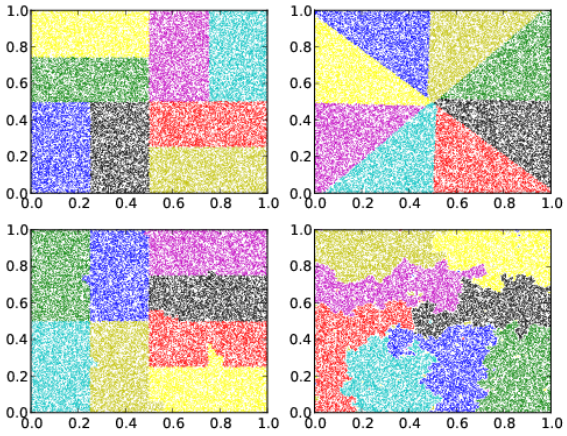

Fig. 4. Particle distribution for 2^{16} particles on 8 processors. Particles are colored according to processor assignment.

qualitative features of the partitions are the same. In particular, we find that the Metis partition is poor with disconnected sub-domains when using particle neighbors as the edges. Using the cell indexing structure as an implicit mesh is an effective way to overcome this limitation. An added advantage is that the graph is now invariant to the number of neighbors for a given particle and the resulting decomposition is better as can be seen in Fig. 5. The choice of cell size is important to reduce the time taken for the decomposition as seen in Fig. 6 This suggests a trade-off between increasing the cell size for faster partitioning times vis-a-vis load balance across processors.
The partitioning times for the geometric algorithms (RCB, RIB and HSFC) are shown in Table. I. The table shows that the RCB is the fastest method for this problem although the actual times are comparable.

B. Dam break with an obstacle

The dam-break problems are an important area of application of SPH. In these problems, an incompressible fluid moves under the influence of gravity in an open vessel. The fluid may encounter obstacles in it's path as for example, a wave striking a lighthouse tower. The challenge is to capture the large deformations of the fluid free surface as a result of

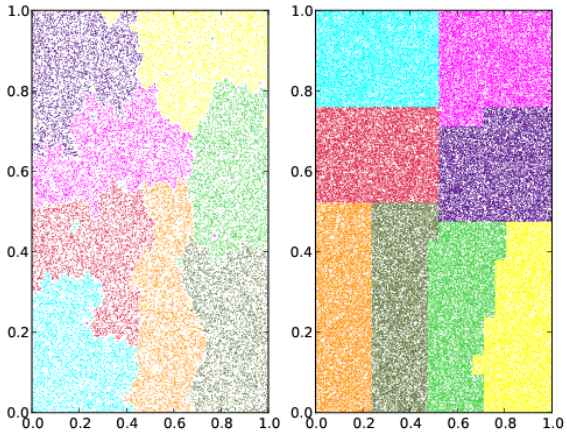

Fig. 5. Graph partitioning using particle neighbors (left) versus cell neighbors (right) using Metis. Notice that using the cell indexing structure produces a better distribution without disconnected sub-domains.

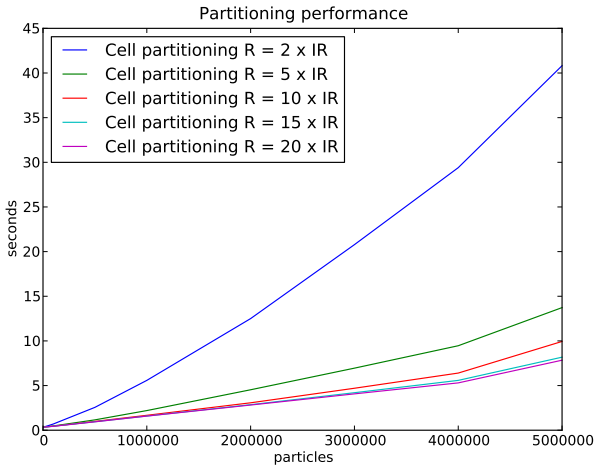

Fig. 6. Partitioning times for Metis using cells as the number of particles is increased. Increasing the cell size results in faster partitioning times.

N_p	RCB	RIB	HSFC
1024	0.0185	0.0124	0.02
2048	0.0187	0.0112	0.010
4096	0.0235	0.0268	0.023
8192	0.0334	0.0523	0.033
16384	0.061	0.0711	0.064
32768	0.115	0.1133	0.119
65536	0.301	0.286	0.286
131072	0.48	0.4894	0.49
262144	1.03	1.112	1.144
524288	2.34	2.283	2.380
1048576	4.8	5.132	5.115

TABLE I
PARTITION TIMES FOR THE BOX TEST USING THE GEOMETRIC PARTITIONERS.

interactions with solid objects. The schematic of the problem we consider is shown in Fig. 7 Representative plots of the

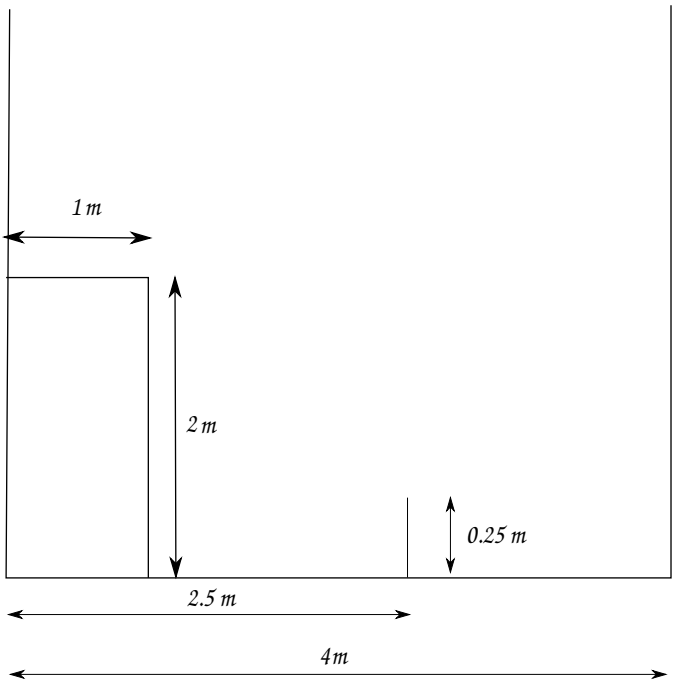

Fig. 7. Schematic for the dam break problem.

distributions produced by the RCB, RIB and HSFC algorithms for 8 processors and 300000 particles are shown in Fig. 8, Fig. 9 and Fig. 10, in which the particles are colored according to the processor assignment. We can see that the qualitative features of the box test are reproduced in this test. We notice that RCB and HSFC algorithms produce disconnected regions in the later stages of the simulation.

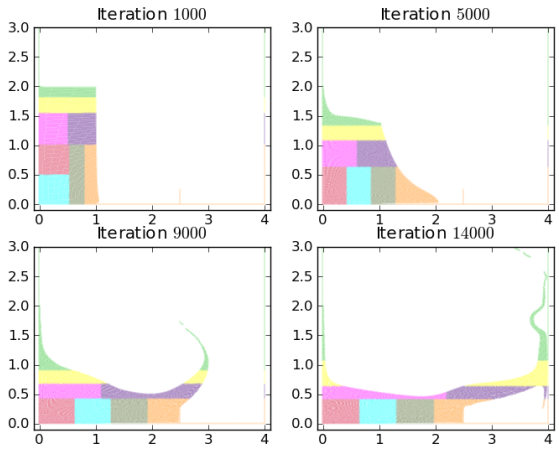

Fig. 8. Distribution for the dam break problem with 8 processors using RCB.

Figure. 11 shows the number of particles exchanged per iteration using the three algorithms for this problem. We

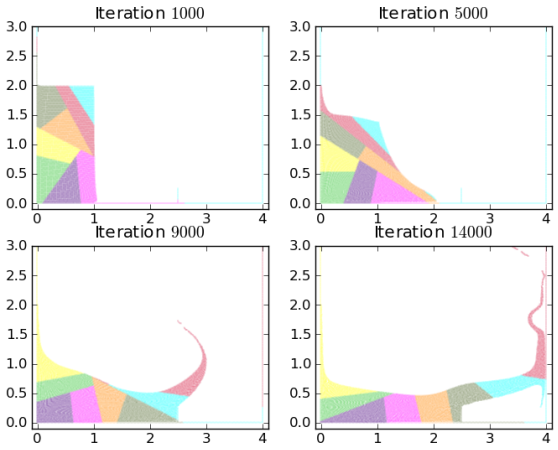

Fig. 9. Particle distribution for the dam break problem with 8 processors using RIB.

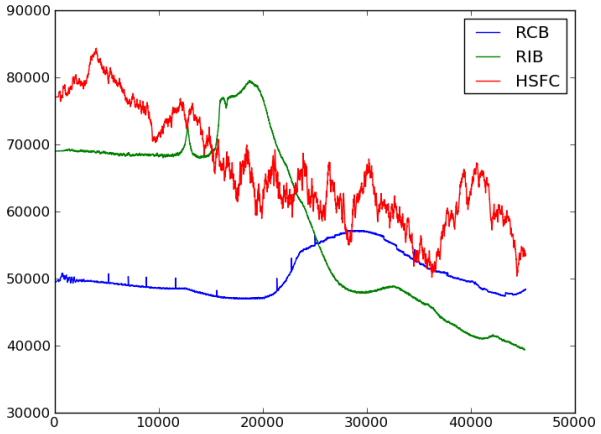

Fig. 11. Number of particles exchanged per iteration for the dam break problem with 8 processors.

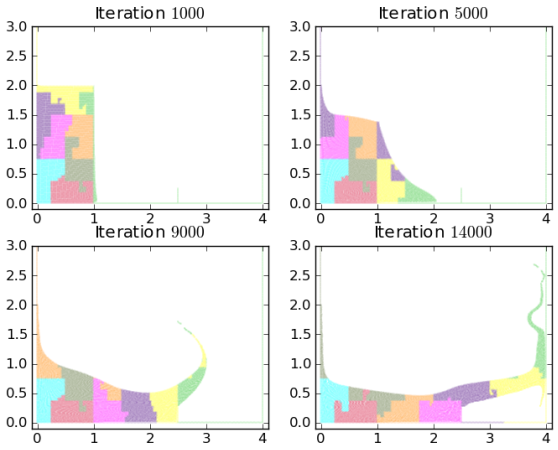

Fig. 10. Particle distribution for the dam break problem with 8 processors using HSFC.

see that initially, RCB produces the best partition since the geometry is axis aligned at this point. With the evolution of the dam front and subsequent collision with the obstacle, the communication overhead of the RIB algorithm is the least. This behavior can be understood with the recollection that the RCB and HSFC algorithms produced disconnected sub domains after the water column hit the obstacle (lower subplots in figures (8) and (10)).

V. Conclusion

We tested different dynamic load balancing algorithms for the SPH particle method where the number of particles is the metric for a balanced workload across. The algorithms tested were the Recursive Coordinate Bisection (RCB), Recursive Internal Bisection (RIB), Hilbert Space Filling Curve (HSFC) and the serial graph partitioning algorithm of Metis.

For the box test (Sec. IV-A), graph partitioning using Metis produced disconnected sub-domains when using the particle neighbors to define the edges of the graph. The quality of the Metis graph partitioning can be improved using the underlying cell indexing structure. The cell size must be chosen appropriately to achieve a good load balance and reasonable running times for the algorithm.

Geometric partitioners are a natural choice for particle methods as the input to these algorithms are the particle coordinates or cell centroids. For the dam break problem with an obstacle, we found that the RIB method produced consistently better partitions than the RCB and HSFC algorithms. In particular, the RCB and HSF algorithms produce disconnected sub domains when the domain is arbitrary leading to large communication overhead and particle re-distribution.

References

[1] D. Price, "Smoothed particle hydrodynamics and magnetohydrodynamics," *Journal of Computational Physics*, vol. 231, pp. 759–794, 2012.

[2] J. Monaghan, "Smoothed particle hydrodynamics," *Reports on Progress in Physics*, vol. 68, pp. 1703–1759, 2005.

[3] Volker Springel, "Smoothed Particle Hydrodynamics in Astrophysics," *Annual reviews of Astronomy and Astrophysics*, vol. 48, pp. 391–430, 2010.

[4] J. Monaghan, "Smoothed Particle Hydrodynamics and Its Diverse Applications.," *Annual Review of Fluid Mechanics*, vol. 44, pp. 323–346, 2012.

[5] Angela Ferarri and Michael Dumbser and Eleuterio F. Toro and Aronne Armanini., "A new 3D parallel SPH scheme for free surface flows.," *Computers and Fluids*, vol. 38, pp. 1023–1217, 2009.

[6] Moncho Gomez-Gesteria and Benidict D. Rogers and Robert A. Dalrymple and Alex J. Crespo, "State-of-the-art of classical SPH for free surface flows.," *Journal of Hydraulic Research*, vol. 48, pp. 6–27, 2010.

[7] John Biddiscombe and David Le Touzé and Francis Leboeuf and Jean-Christophe Marongiu and Guilaume and Jérôme Soumagne, "Efficient Parallelization strategy for the SPH method.," 2015.

[8] Karen Devine and Erik Boman and Robert Heaphy and Bruce Hendrickson and Courtenay Vaughan, "Zoltan Data Management Services for Parallel Dynamic Applications," *Computing in Science and Engineering*, vol. 4, pp. 90–97, 2002.

[9] Erik Boman, Karen Devine, Lee Ann Fisk, Robert Heaphy, Bruce Hendrickson, Vitus Leung, Courtenay Vaughan, Umit Catalyurek, Doruk Bozdag, and William Mitchell., "Zoltan home page," 1999.
[10] J. Monaghan, "Why Particle Methods Work," *SIAM Journal of Scientific and Statistical Computing*, vol. 3, pp. 422–433, 1982.
[11] J. Monaghan, "Smoothed particle hydrodynamics," *Reports on Progress in Physics*, vol. 68, pp. 1703–1759, 2005.
[12] J. Monaghan, "Simulating free surface flows with SPH," *Journal of Computational Physics*, vol. 110, pp. 399–406, 1994.
[13] Allen, M. P. and Tildesley D.J., *Computer Simulation of Liquids* . Clarendon Press, 1989.
[14] George Karypis and Vipin Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs.," *SIAM Journal on Scientific Computing*, vol. 20, pp. 359–392, 1999.
[15] F. Fleissner and P. Eberhard, "Load balanced parallel simulation of particle-fluid DEM-SPH systems with moving boundaries.," *Jon Von Neumann Institute for Computing*, vol. 38, pp. 37–44, 2007.
[16] Volker Springel, "The Cosmologhical simulation code GADGET-2," *Monthly Notices of the Royal Astronomical Scociety*, vol. 364, pp. 1105–1134, 2005.
[17] Youssef M. Marzouk and Amhed F. Ghoniem, "K-means clustering for optimal partitioning and dynamic load balancing for parallel hierarchial N-body simulations," *Journal of Computational Physics*, vol. 207, pp. 493–528, 2005.