51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition
07 - 10 January 2013, Grapevine (Dallas/Ft. Worth Region), Texas

AIAA 2013-0370

Parallel Eulerian-Lagrangian Method with Adaptive Mesh Refinement for Moving Boundary Computation

Chih-Kuang Kuan[1] and Jaeheon Sim[2]
University of Michigan, Ann Arbor, MI, 48109

Ez Hassan[3]
Air Force Research Laboratory, Wright-Patterson Air Force Base, OH, 45433

and

Wei Shyy[4]
University of Michigan, Ann Arbor, MI, 48109
Hong Kong University of Science and Technology, Kowloon, Hong Kong

In this study, we present a parallelized adaptive moving boundary computation technique on distributed memory multi-processor systems for multi-scale multiphase flow simulations. The solver utilizes the Eulerian-Lagrangian method to track moving (Lagrangian) interfaces explicitly on the stationary (Eulerian) Cartesian grid where the flow fields are computed. Since there exists strong data and task dependency between two distinct Eulerian and Lagrangian domain, we address the decomposition strategies for each domain separately. We then propose a trade-off approach aiming for parallel scalability. Spatial domain decomposition is adopted for both Eulerian and Lagrangian domains for load balancing and data locality to minimize inter-processor communication. In addition, a cell-based unstructured parallel adaptive mesh refinement (AMR) technique is implemented for the flexible local refinement with efficient grid usage and even-distributed computational workload among processors. The parallel performance is evaluated independently for the Cartesian grid solver and sub-procedures in cell-based unstructured AMR. The capability and the overall performance of the parallel adaptive Eulerian-Lagrangian method including moving boundary and topological change is demonstrated by modeling binary droplet collisions. With the aid of the present techniques, large scale moving boundary problems can be effectively computed.

I. Introduction

INTERFACIAL flows having moving boundaries and interactions between different phases are challenging in engineering design and operation phases. For example, the draining and sloshing motion of liquid fuel under microgravity in spacecraft have critical impact on its center of mass, and thus the vehicle dynamics control; It requires significant efforts in developing suitable computational techniques to address such flows.[1, 2] Interfacial dynamics also plays an important role in the process of liquid fuel droplets collision and atomization.[3] With the advance of numerical methods, we are able to predict interfacial flow in moderate conditions. However, when interfaces evolve into multi-scale features, computation mesh resolution are highly demanding to support adequate accuracy, and computation is inevitably expensive. The numerical representation of the moving interface either implicitly or explicitly requires additional computational resources than application of single-phase flow. Distinct physical properties across the interface consequently require specific treatment of material jump. Beside, modeling

[1] Graduate Student, Department of Aerospace Engineering, University of Michigan; AIAA Student member.
[2] Post-doctoral Research Fellow, Department of Aerospace Engineering, University of Michigan; AIAA member.
[3] Research Aerospace Engineer, AFRL/RQHP; AIAA member.
[4] Provost & Chair Professor, Department of Mechanical Engineering, Hong Kong University of Science and Technology; also Adjunct Professor, Department of Aerospace Engineering, University of Michigan; AIAA Fellow.

interfacial dynamics such as phase change or surface tension is necessary to realize actual interfacial flow field as well. In summary, multiphase flow computation with moving boundaries is very challenging and computationally expensive due to its multiple time/length scales, tracking of moving boundaries, and interfacial condition like surface tension and discontinuous properties. It extends the requirements of computational power and grids resolution beyond single-processor capabilities.

In such cases, parallel computing can alleviate this burden by distributing computational loads among multiple processors thus allowing the simulation of practical multi-scale flows. On the other hand, adaptive mesh refinement technique can significantly ease computational load by improving resolution adaptively in regions of interests such as interfaces and region with large flow variation. The 3-D adaptive Eulerian-Lagrangian method developed by our group[4-6] showed outstanding accuracy prediction of multiphase flow in serial computations. In our recent work, we studied domain decomposition strategies for Eulerian-Lagrangian method, and tested simple cases without moving boundary at fixed initial grid condition.[7] In present study, we advance our capability to a parallel, adaptive moving boundary tracking method. Parallelisms are applied on the Eulerian and Lagrangian domains, and a cell-based unstructured AMR technique is introduced to adapt the grid dynamically in a parallel manner while balancing the loads between the processors.

Eulerian-Lagrangian method is a dual-domain interface tracking techniques. It utilizes a separate set of moving (Lagrangian) interface mesh on a stationary (Eulerian) grid used to resolve the flow fields. It tracks interface explicitly, and provides outstanding in-cell interface resolution when compared with Eulerian methods such as volume of fluid method and level-set method.[4, 8, 9] Because of explicit interface representation, it provides accurate prediction on morphology-related variables such as surface tension. The Eulerian-Lagrangian method presented in this work use continuous interface method to model material jump around fluid-fluid interface and sharp interface method to accommodate no-slip boundary condition.[6] Besides, it has a dynamic contact-line-force algorithm to model moving fluid-fluid interfaces around arbitrary solid boundaries. This Eulerian-Lagrangian method has been sucessfully applied to many practical engineering problems, such as binary droplet collision, cryogenic fuel sloshing in spacecraft fuel tanks, and interfacial instability.[4-6] Although it is effective approach for interfacial computation, inherently it incurs difficulties toward a parallel implementation. On the perspective of parallelization, it has two features that need to be considered: ratio of computaiton load on Eulerian and Lagrangian data and strong data dependency between Eulerian and Lagrangian domain.

Parallel implementation for Eulerian domain is similar to traditional single-phase flow solvers with an additional transport equation of material scalar variable. Several large-scale geological simulators using VOF method have been successfully applied on distributed memory machines.[10] For studies related to Cartesian grid approaches, Sussman presents a parallelized Cartesian grid solver using coupled level-set/VOF method for flow in general geometries.[11] A parallel sharp-interface method for large scale moving boundary problems in fluid mechanics is proposed by Marella.[12] The *Gerris* code, which is an open source parallel Navier-Stokes solver using VOF method, has performed jet atomization simulation in parallel with octree adaptive refinement method.[13] In general, parallel multiphase flow solvers using the Eulerian method have been well-developed. On the other hand, Eulerian-Lagrangian method in parallel computation has not been studied widely yet. Darmana et al. introduced parallelisms for Euler-Lagrange model aiming for tracking rising bubbles in columns.[14] Mirror domain technique is proposed to decompose Lagrangian particles for load balance while the Eulerian domain is partitioned along z-axis uniformly. The study of parallel Eulerian-Lagrangian method is still constrained to limited cases with simple geometries far from practical engineering applications. Major difficulty in the parallelization of the Eulerian-Lagrangian method is that the computational loads distributed on Eulerian and Lagrangian domains are task-dependent. This characteristic brings complexity to parallelism and communication. As a result, in the current work, possible parallelisms for Eulerian-Lagrangian method are discussed and then spatial domain decomposition is chosen for both domains with the concern of data locality and parallel scalability.

Besides parallel computing implementation, adaptive mesh refinement technique can effectively save computation cost by adjusting space between the computational grid points while maintaining adequate solution. It dynamically refines or coarsens grid according to the desired required grid resolution. Dynamic grid adaption provides efficient management of grid resolution, but incurs extreme challenges in parallel implementation, especially in the aspects of load balance and distributed storage of entire grid. Various strategies have been proposed for the design of parallel AMR. They can generally be categorized by the range of adaptation and the data structure; block-AMR, cell-based tree AMR, and cell-based unstructured AMR. The performance of these approaches is determined either by its inherent data structure or the scale of problems that AMR algorithm are applied on. Moreover, the data structure adopted also affects complexity of the algorithm and domain decomposition strategies.

The cell-based tree AMR utilizes quad-tree or oct-tree data structure to accommodate hierarchy of sub-grids.[12, 15, 16] The tree structure forms a cell-based object containing information of coarsest to finest level of cells for desired spatial resolution and parent-children relationship. It is relatively easy to parallelize if the decomposition of refinement/coarsening procedure is based on the coarsest level grid due to its hierarchy structure. A parallel sharp interface fix-grid method for moving boundary problem was developed using tree-based local refinement by Marella.[12] Agbaglah et al. decomposed the computational domain based on the graph of the coarsest level cells with weighting function to adjust load balance for dynamics AMR operation at run time.[15] Burstedde et al. presented library *p4est*, which is a scalable algorithm for parallel adaptive mesh refinement based on the concepts of forest of oct-trees by connecting the *z*-shaped space-filling curve between oct-trees.

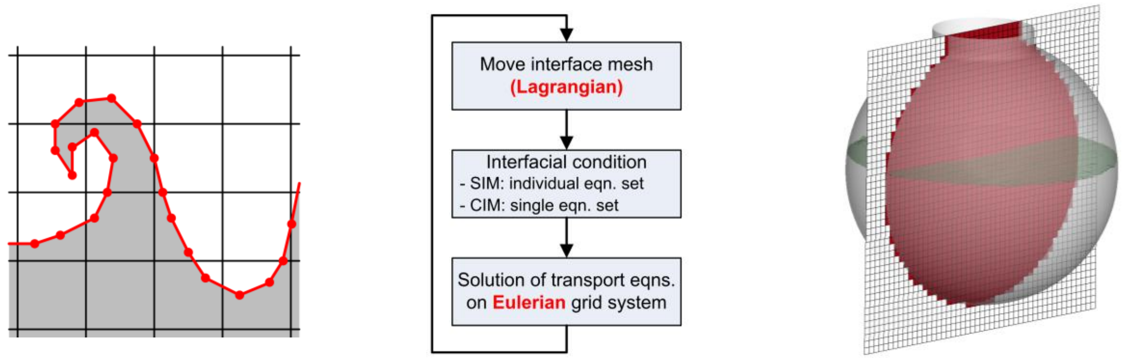
Although the cell-based tree AMR provides flexible refinement region and organized data structure, the computational speed is degraded due to frequent pointer traveling in-and-out of the tree hierarchy when searching for data. Instead, the block-based AMR applies new refinement level of grid block upon regions where higher resolution is demanded. Each individual block is typically a uniform Cartesian grid, which is highly structured. The entire domain is composed by overlapped multiple blocks and domain decomposition is based on block units. At runtime, refinement operation can be applied to specific blocks or new blocks, and they can be overlapped with old, coarser blocks.[17-20] In general, adaptive blocks could be non-overlapping. The block-tree type AMR, developed by Powell et al., refines a set of grid block by bisecting a parent block in each coordinate direction to generate children blocks with higher level refinement[21]. Using tree-like block AMR, they demonstrate good scalability on thousands of CPU cores for MHD equations solvers. On the other hand, a patch-block AMR nests finer blocks over coarser blocks. One of the most known libraries using patch block AMR is *SAMRAI* (Structured Adaptive Mesh Refinement Application Infrastructure). Gunney et al. discuss several methods for parallelizing the clustering algorithm proposed by Berger and Oliger and compared performance for a model SAMR problem on platform up to 16K processors of the BG/L system.[18] *SAMRAI* is adopted by many research codes such as *IBAMR*, an immersed boundary solver developed by Griffith.[22] *PARAMESH* library using block-structured AMR developed by MacNeice et al.[19] has been utilized by Zuzio and Estivalezes[23] for the simulation of two phase interfacial flow. In general, block-structure AMR has merits of better cache optimization due to structured data format. Their communication overhead may be less than cell-based tree AMR due to regular partition boundaries. Moreover, block-based AMR over-refines the grid so that adaption does not need to be executed as frequently as cell-based AMR. However, this feature also increases the amount of data storage and computational time spending on over-refined region, and data interpolation between overlapping blocks requires additional communication over processors.

A block AMR or cell-based tree AMR may tend to waste data storage, and thus computing power, by over-refining meshes in blocks or keeping all hierarchy information from the coarsest- to the finest-cells. A cell-based unstructured AMR, which has no hierarchy information, can be alternative due to its flexibility of adaptation and compact data storage. This approach shows outstanding performance in serial computations.[4-6] However, the development of scalable parallel AMR for these methodologies is challenging especially at large number of CPU cores because of explicit storage of global connectivity and dynamic behavior of indexing. Choices of domain decomposition approach may be limited due to the absence of systematic grid structure and unstructured data packing. Using heuristics partition libraries is a relatively popular choice to accomplish the partitioning work of unstructured AMR.[24] *ParFUM* and is a typical example in this category.[16, 25] In terms of performance, cell-based unstructured AMR has flexible refinement regions and better mesh efficiency especially for problems with complex moving boundaries. It has consistent data fetching rate and no level-level interpolation, which insure field equation solvers has performance independent of refinement level. Heuristic partition libraries for load balance avoid racing at all kinds of situations is favorable. The current work adopting cell-based unstructured AMR method with the proposed parallel re-mesh algorithm can satisfy demands of parallel scalability of Eulerian-Lagrangian method.

The present study is organized as follow. First, we briefly introduce numerical algorithm of the 3-D adaptive Eulerian-Lagrangian method. The interactions between Eulerian and Lagrangian domain during computation are discussed to highlight the design consideration and then parallelisms are proposed. Parallel implementation for cell-based unstructured adaptive mesh refinement is detailed. We analyze parallel performance by considering speedup of flow solver and AMR separately, and then perform a binary droplet collision to study overall performance of parallel, adaptive Eulerian-Lagrangian method.

II. Eulerian-Lagrangian Methods

In the marker-based 3-D adaptive Eulerian-Lagrangian method, the bulk flow variables are solved on the Eulerian grid, whereas the interfaces, separating different phases, are handled by Lagrangian surface meshes. Figure 1 shows an illustration and the solution procedure of the current approach. The present Eulerian-Lagrangian method is summarized in the following sections, and the detailed numerical method can be found in literatures.[4, 5, 9]


Move interface mesh
(Lagrangian)
Interfacial condition
- SIM: individual eqn. set
- CIM: single eqn. set
Solution of transport eqns. on Eulerian grid system

(a) (b) (c)

Figure 1. The Eulerian-Lagrangian method. (a) Interface representation and tracking using moving meshes (red) on the stationary Cartesian grid. (b) Solution procedures. (c) Illustration of the present numerical approach: liquid interface in a spherical container. Field equations are solved on Cartesian cells tagged in red color. Fluid interfaces are represented by moving Lagrangian mesh (green), and solid boundary (gray) is embedded. Boundary condition is enforced by the ghost cell method.

A. Governing Equations

On Eulerian domain, we solve mass, momentum and energy conservation equations for incompressible Newtonian fluid. We account interfacial dynamics as source terms at the RHS of momentum and energy equation. They are momentum forcing term, $\mathbf{F}_f$, accounting the effect of surface tension of fluid interfaces, forcing function $\mathbf{F}_s$ describing no-slip condition on the solid interfaces, and an energy source term, Q_f , to account latent heat effects around fluid interface. Here, $\mathbf{V}$ is the velocity vector, and ρ , μ , and p is the density, viscosity, and pressure, respectively.

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{V}) = 0 \tag{1}$$

$$\frac{\partial (\rho \mathbf{V})}{\partial t} + \nabla \cdot (\rho \mathbf{V}\mathbf{V}) = -\nabla p + \frac{1}{Re} \nabla \cdot \mu (\nabla \mathbf{V} + \nabla^{\mathrm{T}} \mathbf{V}) + \frac{1}{Fr} \rho \mathbf{g} + \frac{1}{We} \mathbf{F}_f + \mathbf{F}_s \tag{2}$$

$$\frac{\partial (\rho C T)}{\partial t} + \nabla \cdot (\rho C T \mathbf{V}) = \frac{1}{RePr} \nabla \cdot (K \nabla T) + Q_f \tag{3}$$

Here, variables are non-dimensionalized by a characteristic velocity ($\mathbf{V}_{ref}$) and reference length scale (L_{ref}), standard gravity (g_0), and liquid material properties (density ρ_l , viscosity μ_l , surface tension σ , heat capacity C_l, and thermal conductivity K_l). The Reynolds, Froude, Weber, and Prandtl numbers in Eq. (2) and (3) are defined as, $Re = (\rho_l \mathbf{V}_{ref} L_{ref}) / \mu_l$, $Fr = \mathbf{V}_{ref}^2 / (\mathbf{g}_0 L_{ref})$, $We = (\rho_l \mathbf{V}_{ref}^2 L_{ref}) / \sigma$, and $Pr = (C_l \mu_l) / k_l$.

The governing Eqs. (1)-(3) are solved using finite-volume projection method on a staggered grid. The intermediate velocity field is computed first, and then, projected onto a divergence-free space to satisfy the mass conservation equation. The convection term is discretized using a 3rd order ENO scheme in space and a 2nd order Runge-Kutta integration in time. The central difference scheme and Crank-Nicholson method are implemented for the viscous term.

At the vicinity of interfaces, we obtain indicator function $I(\mathrm{x})$, a discrete form of the Heaviside step function, by integrating the 1-D form of discrete Dirac delta function δ_h from liquid to gas phase as shown in Eq. (4). The approximate Dirac delta function is calculated over finite thickness of four cell widths. The indicator function has a value of 0.5 at the interface location, and varies from zero to one smoothly across the interface. This indicator function map discontinuous material properties into a continuous form that we can use a single set of equations to describe all fluid phases in the domain with smooth-out material properties across the interface. This approach requires distance information from an Eulerian point to Lagrangian interface, and thus the information is transferred from Lagrangian to Eulerian domain.

$$\mathrm{I}(\mathbf{x}) = \mathrm{H}\left(r = \mathbf{n}\cdot\left(\mathbf{x}-\mathbf{X}\right)\right) = \int_{-\infty}^{r} \delta_h(h)dh \tag{4}$$

The smoothed fluid properties are computed by indicator function in Eq. (5). Here, φ can be any material property such as density, viscosity, and conductivity.

$$\varphi = \varphi_2 + \left(\varphi_1 - \varphi_2\right)\mathrm{I} \tag{5}$$

B. Interface Representation and Tracking

The interface is represented by markers and elements with connectivity information. The geometrical structure is established via line-segments in two-dimensional and triangles in three-dimensional domains. The marker locations, denoted by $\mathbf{X}$ in Lagrangian frame, are updated from the velocities at its location, $\mathbf{V}(\mathbf{X})$, in Eq. (6).

$$\frac{\partial \mathbf{X}}{\partial t} = \mathbf{V}(\mathbf{X}) \tag{6}$$

Fluid interfaces use the computed flow solution field to obtain the marker velocities as shown in Eq. (7). In this equation, the discrete Dirac delta function, $\delta_h(\mathbf{x}-\mathbf{X})$, is employed for converting the Eulerian velocity field, $\mathbf{V}(\mathbf{x})$, to Lagrangian marker velocities, $\mathbf{V}(\mathbf{X})$. The interface velocity is a function of fluid velocity and mass transfer rate $\dot{m}_f$ in case of phase change. The update of markers' location involves Eulerian-to-Lagrangian data communication.

$$\mathbf{V}(\mathbf{X}) = \int_V \mathbf{V}(\mathbf{x})\delta_h(\mathbf{x}-\mathbf{X})dV - \frac{\dot{m}}{\rho_f} \tag{7}$$

C. Interfacial Dynamics Modeling

The fluid interfaces are modeled using the continuous interface method, which involves smoothing variations in material properties and the influence of surface tension, mass, and heat transfer. The surface force computation in Eq. (8), where σ is the surface tension and κ is the curvature of interface, is applied as a source term in the momentum equation, Eq. (2).

$$\mathbf{F}_f = \int_{\Gamma(t)} \sigma\kappa\delta_h\left(\mathbf{x}-\mathbf{X}\right)d\Gamma \tag{8}$$

The latent heat contributing to phase change is computed by Eq. (9), and where $\dot{m}_f$ is mass transfer per unit area across interface due to phase change and L is the latent heat.

$$Q_f = \int_{\Gamma(t)} \dot{m}_f L\delta_h\left(\mathbf{x}-\mathbf{X}\right)d\Gamma \tag{9}$$

Both surface tension and the latent heat of the interface due to phase change are projected from a Lagrangian quantity ($\mathbf{X}$) to an Eulerian quantity ($\mathbf{x}$) via the approximate discrete Dirac delta function, $\delta_h(\mathbf{x}-\mathbf{X})$. Details on numerical computation for surface tension, mass and heat transfer terms can be found in.[5, 6]

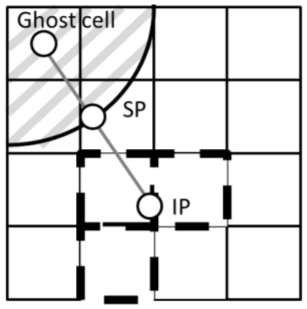


Figure 2. Sharp interface method. Boundary conditions are enforce through reconstruction of solution in ghost cells using linear interpolation of image point (IP) to impose boundary condition

On the other hand, solid interfaces are modeled using the sharp interface method by reconstructing solution fields around an interface to enforce the given boundary condition in the Eulerian Cartesian grid. The velocity, pressure and temperature reconstruction in a ghost cell is extrapolated from fluid side along the local norm of interfaces (Figure 2). The accuracy of solid boundary condition is directly related to the accuracy of reconstruction of solution fields at ghost cells.

Eulerian and Lagrangian domains have frequent data exchange through the delta function. The direction of data communication is either Eulerian-to-Lagrangian or Lagrangian-to-Eulerian depending on the task of computation. Strong spatial data dependency between Eulerian and Lagrangian domains implies data locality is a primary factor on the performance of parallel computation.

III. Parallelization Aspects of Eulerian-Lagrangian Method

The implementation of parallel Eulerian-Lagrangian method should provide substantial computational power with scalability, compact data structures and efficient communication methodologies. For the purpose of portability, this parallelization is designed for distributed memory multiprocessors architecture with MPI standard for communication. Third party libraries, PETSc[26] and HYPRE[27] had been implemented for solving linear system of equations on Cartesian grid in previous study.[7] In present, we concentrate on parallel moving boundaries tracking with cell-based unstructured adaptive mesh refinement technique.

By some testing runs of current Eulerian-Lagrangian solver in serial manner, we observe that solving field equations on Eulerian domain usually costs more than 60% of wall-clock time, and Lagrangian computation takes the rest. As a result, load balance on Eulerian computation is priority, but appropriate decompositions of Lagrangian computation is required to have favorable parallel efficiency. However, this is not easy to achieve because Lagrangian mesh does not uniformly dwell in Eulerian domain. Our goal here is to find a trade-off between load balance of Lagrangian computation and communication cost.

A. Eulerian Domain Decomposition

Among many approaches for spatial decomposition of a graph, the most popular methods are space filling curves method and heuristic method. A space-filling curve traverses N-dimensional graph and maps it bijectively to a one-dimensional array. The ordered cell group is split into *p* parts, where *p* is the total number of processors to satisfy the load balance. However, minimizing the edge cuts, that is, the size of partition boundary requiring information exchange, is not an objective of the space-filling curve method. On the other hand, graph partitioning packages, such as ParMETIS which uses a k-way multilevel heuristic approach usually provide satisfactory load balance and fewer edges cut compared with the space-filling curves method[28]. In this study, the ParMETIS is implemented as the partitioning tool.

Figure 3(a) shows an illustration of Eulerian domain decomposition. This experiment models sloshing behavior of cryogenic fuel in the liquid hydrogen tank on Saturn V/S-IVB rocket. We use dynamic material tags to distinguish gas, liquid, and solid phase as shown in Figure 3(b). Because ghost cells are not evenly distributed among partitions, the computational load of boundary condition is not balance. However, the computational time of this operation contributes very small share of total run-time, we do not perceive overhead due to it. Between adjacent partitions, we build overlapping layers of cells. The overlapping zone serves as a buffer region for synchronization of information. Suppose that an Eulerian domain Ω has already been decomposed into n non-overlapping partitions Ω_p, which p is the processor ID and n is number of total processors.

$$\Omega = \bigcup_{p=1}^{n} \Omega_p \tag{10}$$

We defien a sub-domain S_p which is the union of partition Ω_p and regions of finite width away from partition Ω_p overlapping with its adjacent partitions.

$$S_p = \Omega_p \cup (\bigcup_{i=1}^{adj} I_{p,i}) \tag{11}$$

Here the $I_{p,i}$ represent the overlapping of sub-domains S_p and its neighbor partition Ω_i . The local grid connectivity is constructed on sub-domain S_p . Note that the overlapping region, $I_{p,i}$, is purely served as ghost cells for storage of data from processor *i* through communication routines and does not involve in computation.

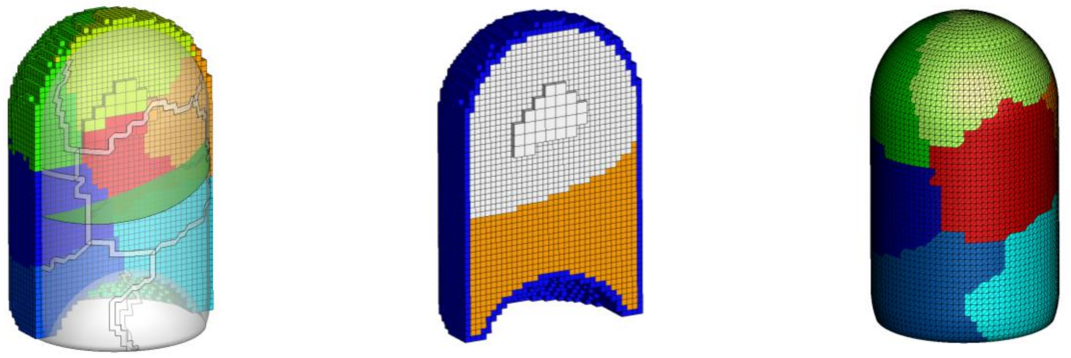


Figure 3. The Eulerian domain decomposition. (a) Partitions are represented in different color. (b) Grid cell material at gas phase (white), liquid phase (yellow), and solid phase (blue). (c) Decomposition of solid geometries according to Eulerian partitions.

B. Lagrangian Domain Decomposition

Three different approaches are considered to achieve Lagrangian decomposition for implementation in the current effort: task parallelism, atomic decomposition, and spatial decomposition. Figure 4 illustrates the application of approaches in the decomposition of a one-dimensional interface denoted by red dots superimposed on a background Eulerian Cartesian grid into four partitions.

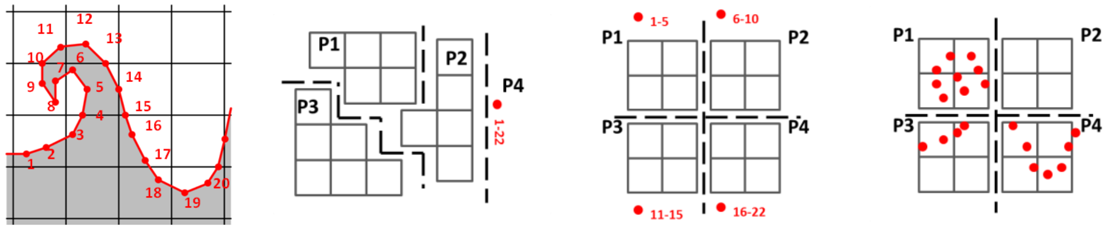


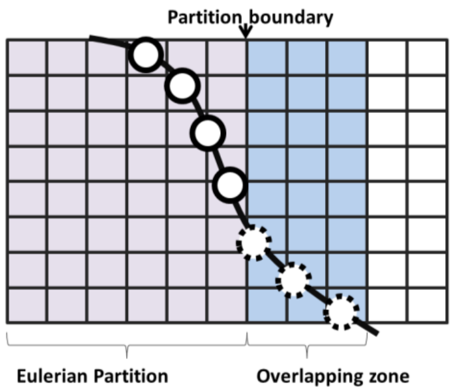
Figure 4. Strategies of decomposition of Lagrangian interface. (a) A 1-D Lagrangian mesh on Eulerian domain. (b) Task parallelism. (c) Atomic decomposition. (d) Spatial decomposition.

Task parallelism reserves specific processors to dealing with Lagrangian data exclusively and using the rest of processors for calculation of Eulerian data. However, according to the procedures of computation, we do not have much concurrency of Lagrangian tasks and Eulerian tasks. The improvement of computational speed will not be desirable. Other challenge is that the optimal arrangement of fair work load among Eulerian and Lagrangian tasks for general problems. The ratio of workload between Eulerian and Lagrangian domain is problem-dependent and varies with the evolution of the interface. Complex management is required to avoid processes idling due to racing among tasks.

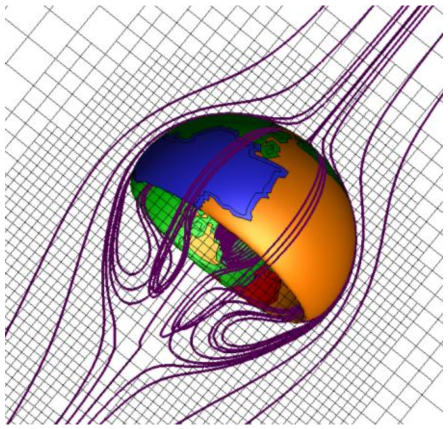
Data parallelism distributes the entirety of Lagrangian data over processors independent of their relationship to background grid to be processed concurrently. Under the data parallelism, one of the choices is to decompose Lagrangian interface using atomic decomposition[29] (Figure 4(c)). Atomic decomposition evenly distributes Lagrangian markers to all processors regardless of their locality. It achieves load balance automatically. However, frequent communication for updating interface information between processors due to low locality is inevitable. Data exchange under atomic parallelism may intensely rely on all-to-all communication that leads to communication bottleneck. Thus deterioration of parallel efficiency is expected to arise from communication overhead.

Unlike atomic decomposition, spatial decomposition spreads the entire Lagrangian interface into sub-domains on the basis of spatial inhabitation of markers. In Figure 4(d), Lagrangian markers encompassed by an Eulerian sub-domain owned by a particular processor are assigned to this processor. This approach takes advantage of data locality because interface information is available in each processor thus mining both communication frequency and data size exchanged. The load balance of the Lagrangian computation may not be satisfied but reasonable scalability is expected in general cases since the cost of pure calculation of Lagrangian data is smaller than cost of flow solver. Besides, the methodology of spatial decomposition of the Lagrangian domain is similar to Eulerian domain, which renders a straightforward implementation. As a result, we select spatial decomposition as the partitioning strategy in current approach.

The Lagrangian domain decomposition procedure is summarized as follows. A marker locality array is used to account for marker inhabitation in which Eulerian partition encompasses this marker. As shown in Figure 5(a), localities of the solid and dashed markers inside a particular Eulerian sub-domain are assigned with this primary processor ID. The collection of partition-own markers, represented by solid circle, is the sub-set of local Lagrangian sub-domains that operations are looping on, and dashed markers in overlapping zone are served as buffers whose information is updated by neighboring processors at the end of each time step. Note that the global connectivity of Lagrangian interface will preserved for repartitioning. Repartitioning is necessary when a marker moves out of its Eulerian sub-domain. In addition, coarsening and refining operations on the Lagrangian interface which change global connectivity requires repartitioning.



(a)

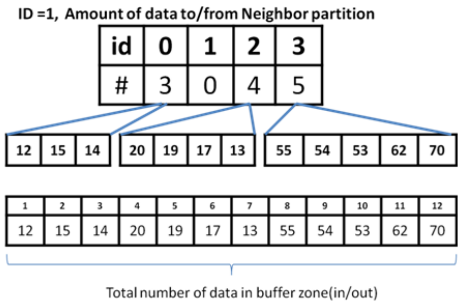


(b)

Figure 5. Illustration of Lagrangian domain decomposition. (a) Lagrangian markers inhabiting in an Eulerian sub-domain constitutes a Lagrangian sub-domain. (b) Streamline of a single rising bubble driven by buoyancy force. Interface is colorized with respect to partitions.

C. Communication Methodology

Data on an overlapping zone of a sub-domain needs to be synchronized with neighboring sub-domains for data consistency. For point-to-point communication, i.e., a sub-domain synchronizing data in an overlapping zone with neighboring partitions, the non-blocking send and receive are used to avoid blocking delay. The sent-out data are concatenated into a data chunk on the sequence of add-up manner. Figure 6(a) illustrates how the data in overlapping zone is grouped in a chunk of continuous memory. This problem is executed with 4 processors and the partitioned domain of processor 1 is adjacent to processor 0, 2 and 3 with 3, 4, and 5 entries in overlapping zone respectively. A temporary array is then created by concatenation of prepared data. The pointer of this array is passed to the subroutine, *MPI_ISEND* to initialize communication process. The procedure of receiving data from neighbor partitions utilize a similar approach by allocating one-dimensional space beforehand for arriving data. An *MPI_WAIT* is placed in the end of initialization of send and receive subroutines to check if procedures are completed correctly. The pseudo code of the communication procedure is shown in Figure 6(b). This design provides possibility of overlapping between communication and computation that decrease communication



```
Do : index0 = 1, total number of processors
    IF NOT neighbor partition, CYCLE
    recv → recv_data(index0); source = index0
    tag1 = processor_id
    CALL MPI_IRECV(recv, # of receive, type, source , tag1, comm.,
    reqs)
    send → send_data(index0)
    tag2 = processor_id;  dest. = sent entry index
    CALL MPI_ISEND(send, # of send    , type, dest.   , tag2, comm, reqs)
Any computation work
CALL MPI_WAITALL(reqs, stat, comm)
```

(a) **(b)**

Figure 6. Communication for the overlapping zone. (a) An example of concatenated array of data in overlapping zone of processor ID 0 on 4 processors computation. (b) Pseudo code of communication routines for exchange data on overlapping zones.

overhead.

The performance of our methodology is evaluated with a one million cells problem. The problem is executed on the NYX machine in the Center of Advance Computing at University of Michigan. The nodes we used on NYX are comprised with Intel Nehalem Xeon E5540 CPU with InfiniBand networking whose latency is about 10^{-6} second. Each Intel Xeon E5540 CPU has 4 cores with 8M Bytes L3 shared cache and total memory available per CPU is 12G Bytes. The averaged data size of velocity exchanged in a process of communication is from 108K to 41 Bytes on 4 to 96 processors. The start-up time represents the preparation and calling of *MPI_ISEND* and *MPI_IRECV* procedures, and the time to finish synchronization stands for time staying on *MPI_WAITALL*. The cost of one velocity data synchronization is on the order of 2×10^{-5} second independent of the number of processors. Our communication strategy has been tested and trivial latency is observed. This communication design is applied to data exchange on the ghost zone of partitions for both Eulerian and Lagrangian data computation (not all-to-all.)

Table 1. Communication cost for velocity filed in 1 million cells problem.

# of processors	4	8	16	64	96
Start-up time [sec]	3.0×10^{-5}	2.4×10^{-6}	2.0×10^{-5}	2.1×10^{-5}	2.1×10^{-5}
Time to finish synchronization [sec]	9.5×10^{-6}	9.7×10^{-6}	9.5×10^{-6}	1.0×10^{-5}	1.1×10^{-5}
Averaged data size (K Bytes)	108	105	84	50	40.5

D. Scaling of Computation and Communication Cost

Parallel performance of procedures are highly dominated by the computation-to-communication ratio. We summarize the computation and communication cost in Table 2 to illustrate how procedures scale with problem size. Here N is the total mesh size of a problem. Procedures types are divided into three categories. Eulerian tasks refer to the computation on the Eulerian frame, i.e., Cartesian grid solver. The leading cost of Eulerian computation is solving the linear sparse matrix of Pressure Poisson equation, which is at most proportional to $O(N^3)$. Communication required for Eulerian tasks are data at a partition's boundaries, which is the size of "surface" and proportional to $O(N^{2/3})$. For Lagrangian tasks, computation scales with surface area $O(N^{2/3})$ and data size of communication between Eulerian and Lagrangian domain is in the order of $O(N^{2/3})$. For adaptive mesh refinement

(AMR) using geometry-based refinement, the computation grows with O($N^{2/3}$), but the data need to exchange may as large as volume of a domain because of the process of constructing global grid connectivity. As a result, for a fix-size problem, Eulerian tasks have higher computation-to-communication ratio than Lagrangian tasks, and AMR has the least computation-to-communication among all. According to this estimation, the parallel performance of sub-procedures of Eulerian-Lagrangian method are highly related with the procedures types.

Table 2. Scaling of computation and communication cost of adaptive Eulerian-Lagrangian method

	Computation	Communication
Eulerian tasks	O(N^2~N^3)	O($N^{2/3}$)
Lagrangian tasks	O($N^{2/3}$)	O($N^{2/3}$)
AMR	O($N^{2/3}$) (geometry-based)	O(N)

E. Solution Procedure

The solution procedures of parallel Eulerian-Lagrangian method are summarized in Figure 7. Note that operations in italic involve both Eulerian and Lagrangian variables. First, the initialization of simulation creates a Cartesian grid to encompass solid interface and refine it adaptively to designated resolution with the identification of ghost cells. The Eulerian domain is then partitioned by ParMETIS, and the Lagrangian domain is spatially decomposed based on marker's locality. In the time integration loop, a marker is updated to new position by a velocity field interpolated from the Eulerian domain and re-meshing may be applied to Lagrangian interface whenever needed. We re-decompose Lagrangian domain and then update fluid properties at overlapping zone of Eulerian sub-domains. The intermediate velocity field is updated by Runge-Kutta/Crank-Nicolson time integration method. If adaption is required, the program inquires AMR routines including parallel mesh generation, Eulerian domain re-decomposition, data migration and generation of new Eulerian and Lagrangian sub-domains. The detail of parallel adaptive mesh refinement is explained in the next section. PETSc and HYPRE are both implemented as the linear solver for the discretized Poisson equation. Once the divergence-free velocity filed is obtained on Eulerian domain, it is projected to Lagrangian markers by Dirac delta function.

Initialization: **Eulerian domain decomposition(ParMETIS)**
Lagrangian domain decomposition

Time integration loop :

Marker movement
Interface Modification: Smooth, refine, coarsen
Lagrangian domain decomposition

Determine cell-centered material and properties

Source term: ***Surface tension and Mass transfer/ Heat flux at fluid interface***

Intermediate velocity U* by RK-CN integration

IF : Dynamic adaptive mesh refinement
Parallel mesh generation
Domain Re-decomposition(ParMETIS)
Data migration
Rebuild Eulerian sub-domain and Lagrangian sub-domain

Pressure Poisson equation ***HYPRE/PETSc***

Corrected velocity $\mathbf{U^{n+1}}$

Energy equation by RK-CN integration

Velocity interpolation to Lagrangian markers

IF: reconstruction needed, do ***Reconstruction of Lagrangian interface***

Figure 7. The solution procedure of parallel Eulerian-Lagrangian method.

IV. The cell-based unstructured parallel AMR

The AMR implemented in cuurent study utilizes adaptive Cartesian grid with cell-by-cell isotropic adaption and unstructured data storage[6]. Figure 8 is a simple two-dimensional demonstration of the unstructured cell-based AMR. Cell 2 and 3 are taged as refiement required and each of them are splitted into 4 equal finer square cells. The originally index “2” and “3” are pointed to one of the four children cells, and other added cells are assigned with new indices. A coarsen operation is then requested to cell 3 resulting a deactivation of cell 5, 6, and 7 with the original cell index “3” pointing to the merged cell. Computation on data of cell 5 to 7 are skipped, but their indices can be reserved with inactive tag. The split and collapse of faces utilize the same mechanism to accommondate dynamic data allocation of our unstructured cell-based AMR method. Note that the unstructured cell-based AMR

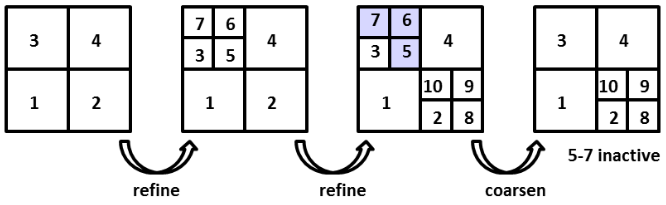


Figure 8. Demonstration of current cell-based unstructured AMR. Splitting one cell with added active cell index to 3 children cells for refinment and merging 4 cells by deactivateing cells index during coarsening.

does not have tree hirachy to pertain a cell’s parent-child releationship. It is pure unstructured grids.

Due to the nature of the unstructured data format, the global connectivity of entire mesh is retained all the time. Such unavoidable memory requirement is challanging for the unstructured AMR method to compute very large problems on distributed memory architecture. Figure 9 gives an estimation of memory cost of our design. N is the size of a problem and p is the number of processors used in a computation. Storing global grid is the leading cost of on system memory. In the current approach, the connectivity arrays storing global grid are cell-face list and face’s side cell pair. The necessary global connectivity and variables occupy 9 percent of the memory requirement in a single processor computaiton, and the rest are dividable data. For example, a 2.0×10^7 mesh size problem demands 1750M bytes memory to accommondate global connectivity in each processor, and the rest of memory usuage are spent on local and dividable variable arrays.

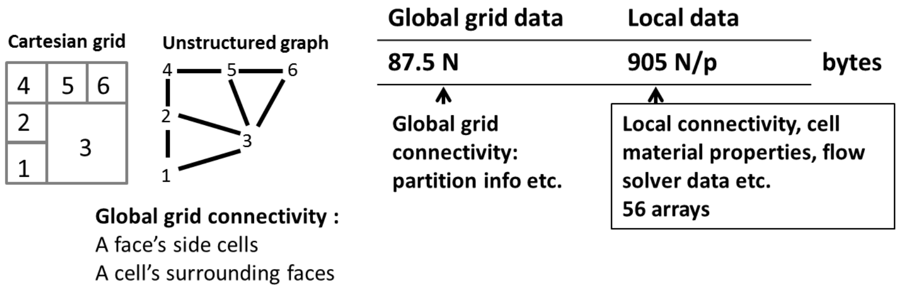

Cartesian grid
4
5
6
2
3
1
Unstructured graph
4
5
6
2
3
1
Global grid connectivity :
A face’s side cells
A cell’s surrounding faces
Global grid data
Local data
87.5 N
905 N/p
bytes
Global grid connectivity: partition info etc.
Local connectivity, cell material properties, flow solver data etc.
56 arrays

Figure 9. Memory cost of Eulerian-Lagrangian method with cell-based unstructured AMR method.

A. Implementation of Parallel Cell-based Unstructured AMR

The primary considerations of our parallel cell-based unstructured AMR are load balance throughout most of dynamic AMR processes and reduction of communication frequency and volume.

As been discussed, parallel cell-based unstructured AMR(UAMR) features flexible refinement regions, better mesh efficiency and load balance regardless the number of processors and mesh size. Although the nature of unstructured AMR implies its remesh process may be more difficult than structured AMR, we are aiming to explore the overall performance of parallel unstructured AMR under the scope of hundreds of processors computation.

The following five steps present the framework of our parallel AMR.

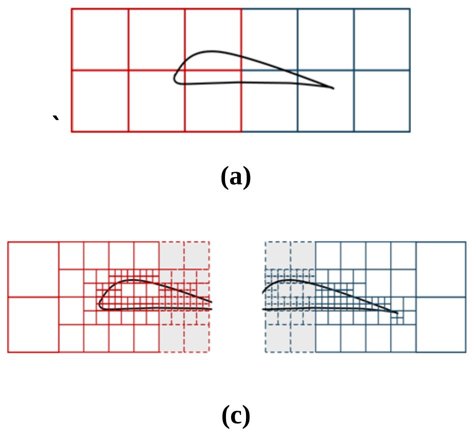
1) Parallel adaptive grid generation on Eulerian sub-domains
2) Rebuild global indices of cells/faces and connectivity
3) Domain re-decomposition of new graph via partition libraries- ParMETIS
4) Data migration across processors
5) Create new Eulerian and Lagrangian sub-domains

The unstructured AMR starts with isotropic grid adapation based on the adaption flag. This process utilizes a paralleized unstructure grid generator which is upgraded from an existing sequential grid generation algorithm. The created sub-graphs in each processor are indpendent upto this step. In other words, a new grid in sub-domain *A* does not have knowledge of the new grid of sub-domain *B*. We then utilize two steps to remesh global grid: an add-up indexes to give global order of the created faces and cells in each sub-domain; and an algorithm to combine connectivity of adjecent sub-grid. Once the global grid is updated, the partitioning libray-ParMETIS, is introduced for domain decomposition. The forth step includes about data redistribution from a processors's original sub-domain to new sub-domain. Once the old sub-domain-owned data set is backed-up, the new Eulerian sub-domains are created and filled in with data stored in data migration arrays.

B. Parallel Grid Generation on Eulerian Sub-domains

The adaptation criteria of cell-based AMR consider both interfacial geometry and flow solution features at local cells. For interface geometry-based adaptation, all cells cut by the interface and five layers of cells around the interface-cut cells are flagged for refinement for smooth mesh size transition. The range of refinement around interface determines frequency of AMR required. With larger range of refinement, interface stays in the finest-level grid longer that ARM frequency may reduce. However, large refinement area may introduce inefficient mesh usage. Cells far away from the interface are adapted based on the solution of flow field. The decision to refine or coarsen a cells is determined by comparing its local value to standard deviations of vorticity or the temperature gradient in entire domain Ω. Cell-centered pressure and temperature and face-centered velocity of refined cells are constructed linearly using information from the surrounding cells. The variables in a coarsened cells are simply averaged using the corresponding cell-centered or face-centered values of the parent cells.

The grid adaption is applied on a partition and its overlapping zones. However, only the mesh generated in a partition are adopted as new mesh. The reason that refinement has to be executed on the overlapping zone is due to recursive refinement procedure (Figure 10(c)). We do not refine a cell having a coarser neighboring cell. The coarser neighboring cell has to be refined first thus current cell can be refined. This constrain results in a scheduling sequence of cell-splitting. Extent the refinement to the overlapping region can guarantee consistent refinement results of two adjacent partitions. On the other hand, coarsening process is applied on a partition with restrictions. For example, a cell is not permitted to have neighbors with a lower and a higher refinement level. Moreover, on partition boundaries, cells dwelling in different partitions are not allowed to collapse into a single cell. In other word, coarsen operation is not executed across partition boundaries.



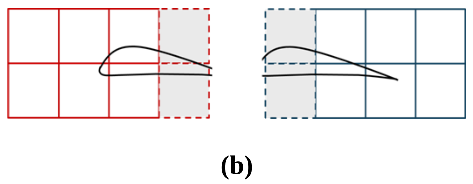


Figure 10. An example of pure geometry-based grid refinement with 2 processors. (a) Partition 1 and partition 2 with an airfoil-shape immersed boundary. (b) Eulerian sub-domain S_1 and S_2 are denoted in red and blue mesh with cells adjacent to partition boundary $\partial\Omega$ having grey coloration, which is treated as an overlapping. (c) Refinement is applied to Eulerian sub-domain. (d) A 'stitching' algorithm corrects and synchronizes the global face indices at partition boundary. The mesh generated in overlapping zone is discarded.

At the beginning of grid refinement, a set of a local grid Z_p is prepared by grouping cells tagged with refinement /coarsen flag and their neighboring cells as shown in Figure 10(b-c). This local grid contains partial grid connectivity of sub-domain S_p in which the tentative modified connectivity due to refining and coarsening operation can be placed. The refinement routine recursively operates over local grid Z_p to generate new cell and faces and write out new graph to Z_p. The coarsening operation merges cells and faces that composes finer levels of cell group by deactivating index pointers of original cell and then adjust grid connectivity. Till this step, load balance of grid generation is achieved in case of even distribution of refinement-tagged cells among all processors. In case of the Lagrangian interfaces encompassed solely at some particular Eulerian sub-domains, refinement operation is applied to those sub-domains alone which induces load imbalance. However, at this worst scenario it is not helpful to adopt other parallelisms such as task (procedural) decomposition because procedure dependency of AMR avoids possibilities of task overlapping. Although the load imbalance of grid generation is inevitable for extreme cases, moderate scalability could still be expected.

C. Global Indices and Connectivity Construction

The new sub-grid of a sub-domain after local mesh generation is independent and is not known by its neighbors, as shown in Figure 10(c). For example, a shared face on partition boundaries is broken due to cell refinement in sub-domain S_1, but the corresponding face in S_2 does not have this index information of new face generated from S_1. However, the new local graph of sub-domain S_1 and S_2 are legislative individual, it is required to coordinate the modified information of this face upon the global graph for consistency. This process of connectivity synchronization is a typical challenge of parallel grid generation. Some strategies approach grid-up of partition boundaries by checking against border faces of partitions sequentially before or after the grid-up of interior sub-domain,[30] which increase serial operation portion for the entire AMR process. Besides, construction of global indexes for new graph and connectivity in parallel is challenging as well. It requires sophisticated algorithms and tedious data exchange at grid generation process. In order to simplify algorithm complexity and maintain efficiency, we separate the indices generation and connectivity process by facilitating two data class containing the modified cells and faces information to accomplish the construction of new global graph Ω''. The sets of data class memorize all modified cells and new generated faces during the refining and coarsening processes. In the data structure of modified cells Mc_p, a collection of broken or merged cells stores the original global index of cell c_k and its sibling cells.

$$\begin{aligned}
&Mc_p = \{c_1, c_2, ..., c_k\}_p, \quad \text{k is the number of modified cell group in processor } p \\
&c_k = [\mathrm{g_i}, \mathrm{s_{3k+1}}, \mathrm{s_{3k+2}}, \mathrm{s_{3k+3}}]^T, \ \text{for spliting operation} \\
&c_k = [\text{-}\mathrm{g_i}, \mathrm{g_{i2}}, \mathrm{g_{i3}}, \mathrm{g_{i4}}]^T, \qquad \text{for merging operation}
\end{aligned} \tag{12}$$

Because sibling cells' global indexes are unknown beforehand, a temporal, sub-domain based indexes as denoted by s_k are assigned to them. In case of cell merge operation, negative signs are symbolically attached to the global index of parent cell g_i, which will be the resulting index representation after merging. The other three or seven children (on 2D and 3D domain) cells are denoted in c_k and will be deactivated. The other data structure called new face information Nf_p accumulates all faces that are generated during refining process in partition Ω_p.

$$Nf_p = \{f_1, f_2, ..., f_m\}_p, \quad \text{m is the number of new faces in processor } p \tag{13}$$

Nf_p holds new faces orientation, face's side cells indexes and its parent face index if generated through splitting a face. In case of faces generated on partition boundaries, a function synchronizing new faces indices on two adjacent partitions. This function goes through all boundary face $f^i_p \in \partial\Omega_p$ and check the 'broken' status of their corresponding face in adjacent partitions. For face pair are both marked in new face data array Nf_p, only the primary processor (lower rank ID) possesses the right to create new global indexes that can avoid duplication of faces.

All-to-all collective communication is used to broadcast Mc_p and Nf_p among all processors and then each processor concatenate collected chunk of Mc_p and Nf_p into contiguous modified global cell-face data structure as below.

$$\begin{aligned} Mc &= (Mc_1, Mc_2, .., Mc_n) \\ Nf &= (Nf_1, Nf_2, .., Nf_n) \end{aligned} \tag{14}$$

Once each processor has a full knowledge of the number of added and merged cells, we assign the global indexes to new cells due to refinement operation in an add-up sequence by looping through Mc array. For example, the number of added cells on a three processors computation is [10 11 9] and the original total size of cells is 50, the new global indexes of new cells are [51:60 61:71 72:80]. Thus, the assembled contiguous indices can be used to update the cell connectivity by renumbering the obtained local arrays. In case of merged cells, inactive tags are assigned to them. Inactive cells and faces do not participate in the new partitioning graph and solution marching process. Face indexes are assigned by a similar add-up manner. A face's front and back cells are updated by the determined global cell indexes. The final procedure to grid-up new global graph is collecting those faces composing a global cell to form a cell's faces list.

Note that after the creation of Mc and Nf by all-to-all collective communication, every processor operates on an identical set of data. As a result, the updating of connectivity of graph is pure serial computation. Moreover, it requires only two messages of communication to accomplish updating of global connectivity with present algorithm.

Performance deterioration due to frequent communication during the adaptive refinement process is avoided by the current approach. It is a fully decoupled method requiring no communication during the remeshing of each sub-domain.[31] Besides, we do not need to grid up the boundary and interior of a partition separately; single time grid generation on each sub-domain is satisfactory.[30] The follow-up correction to the connectivity of global grid is specific concentrated on adaptation-affected regime, which is effective in terms of computation.

D. Domain Re-decomposition

To maintain equal workload among processors after grid adaptation, the new global graph is subjected to re-decomposition. We utilize heuristic partitioning libraries ParMETIS to accomplish this work not only because of the nature of unstructured grid, but also its ability to achieve multiple objectives optimization. Redistribution cost, load balance and the edge-cut can be considered in a partitioning process at the same time. The dynamic domain decomposition starts with a re-ordering process for all active cells that participate in solving the fluid field. Those that have been flagged as inactive global cells and faces due to the coarsening operation will not be grid entities in the distributed sub-domains but only an index pointer in the global graph level. The deactivation strategy is convenient for data structure operation at this stage. A memory adjustment operation will remove all non-active index pointers that are no longer used.

The re-ordering process places a natural index system to active cells of the new graph, and is used to pass the CSR format required at the ParMETIS' input. The weight of a cell is one but promoted to higher value if it is a solid ghost cell to account additional work of boundary condition enforcement.

E. Data Migration and Construction of Eulerian and Lagrangian Sub-domains

ParMETIS returns a partition vector which represents the new processor belonging. Between discard of the original sub-domain Ω_p and construction of new sub-domain Ω''_p, information such as cells' coordinate, level, cell-centered material, pressure and temperature are packed for data migration. Depending on the difference between the original partitions and new partitions, data migration may be communication-intensive. If the AMR are performed at adequate intervals, a good partition library should return partitioning with small change that reduces data redistribution cost. Diffusive scheme of re-partition routines ParMETIS_V3_AdaptiveRepart usually leads to moderate data migration cost based on our experience.

Communication routines for data migration are carefully designed to avoid communication overhead. Cell-centered and face-centered data are packed into two individual user-defined types. As a result, two messages (one for cell-centered and one for face-centered data) are enough to accomplish data migration between two processors. Sub-domains' data need to be transferred to other processors are concatenated into a 1D array and then exchanged by non-blocking send and receive mechanism. Once the redistribution of data is completed, the original Eulerian sub-domain is discarded and then the construction of new Eulerian sub-domain is followed. The Lagrangian sub-domains are built based on new Eulerian sub-domains.

IV. Numerical results

In this section, we first investigate the independent parallel performance for the implanted Cartesian grid flow solver and cell-based unstructured AMR method separately. Then, the overall performance of the present approach will be demonstrated using binary droplet collision with six level of grid refinement. The total mesh size varies from 2.2×10^6 to 2.4×10^6 cells. All numerical experiments exercised are conducted on NYX machine at the University of Michigan, and its specifications have been addressed in the previous section.

A. Cartesian Grid Flow Solver Performance

First, the performance of the Cartesian grid flow solver is evaluated by running the lid-driven cavity flow in three mesh sizes; 1.6×10^5 cells, 1.0×10^6 cells, and 4.0×10^6 cells, without dynamic adaptive mesh refinement. The linear solvers used here is PETSc' conjugate gradient method with Jacobi preconditioner. Since the graph of a linear system of the discretized Poisson equation changes with the number of partitions, the iteration number reaching a constant rounding-error varies with the number of processors used. In order to control the workload spent on solving a linear system on a varying number of processors, executing time of solving Poisson equation is under the basis of the fixed iteration number of the linear solver.

Results show that up to 70 percentage of wall time in a time step is spent on solving the linear system of the discretized Poisson equation. As a result, overall parallel performance is dominant by the efficiency of linear solver, which is controlled by two factors: the Computation-to-communication ratio and shared cache size per CPU. For a fixed-size problem, increasing the number of processor decreases computation-to-communication ratio, and results in the slowdown of speedup. In case of shared cache effect, its impact on the parallel performance is related to the size of working data in solving a linear system. A typical parallel program assigns a fraction of the entire data to each processor. However, the entire data set does not fit into the cache on a single processor execution but a parallel computations executing on the same problem with multiple processors may have working data assigned to each processor that can fit in its local shared cache. In case of a computation having fully cached decomposed working data, a high hit rate and super-linear speedup are achieved.

Figure 11(a)-(c) shows executing time and speedup of three cases of different mesh size. With smallest grid size of 1.6×10^5, the speedup increases gradually with increasing number of processor but slows down on 16 processors due to decreasing computation-to-communication ratio. In the moderate grid size of 1.0×10^6 cells, super-linear speedup due to higher cache hit rate is observed in between 64 to 108 processors and the slowdown of speedup due to decreasing computation-to-communication ratio is found on 140 processors. The last case has 4.0×10^6 cells such that the working data is excessively large with respect to the 8M-Bytes L3 cache on a serial execution, and results in significantly low cache hit rate that overestimates the speedup of multiple-CPU computation. The parallel efficiency gradually ascends to 1.2 at 240 processors. Deterioration of the efficiency arising from the decreasing computation-to-communication ratio is not observed in the scope of 240 processors.

Figure 11(d) gives the parallel efficiency based on number of cells per processor. In general, the efficiency reaches peak roughly at 1.5×10^4 cells per processor and deterioration due to decreasing computation-to-communication ratio is around 10^4 cells per processor. It suggests that parallel efficiency does not benefit from further decomposition of the data. On the other hand, relative low efficiency at larger number of cells per processor is observed because of lower cache hit rate. The highest efficiency happens when working data are fully cached. As a result, the 8M-Bytes shared cache affects the behavior of parallel efficiency with larger problems. Moreover, for working data that are able to be fully cached on a serial execution, the cache effect on parallel efficiency will not be seen as the case of 1.6×10^5 cells. The overall performance of the Cartesian grid flow solver demonstrates favorable efficiency for problems in the scope of several million of cells on Intel Nehalem Xeon architecture. Based on the tests above, the Cartesian solver has best performance at 15,000 to 20,000 cells per processor.

B. Performance of Cell-based Unstructured Adaptive Mesh Refinement

We focus on parallel performance of AMR by conducting both strong and weak scaling studies for current AMR approach. The refinement (or coarsening) is applied on an initial uniform grid by random assignment of adaption flag on every partition. Due to using randomly assignment of adaption flag, run time spent on repartition is small and consequently new partitions do not show significant data redistribution cost. Each partition obtains 2 percentages of cells having refinement flags that produce 14 percentage of total cells increment, and the spatial distribution of applied refinement is arbitrary. In the strong scalability test, we start with original mesh size 8.19×10^6 and then reach about 9.3×10^6 after AMR. Because the data packing of our approach is unstructured and cell-based, number of refinement level in the domain will not affect data-fetching rate and load balance theoretically. Hence, this experiment considered one level of refinement and characteristic of performance is applicable to cases having multiple levels of refinement.

We categorize three primary procedures composing an AMR operation: adaption flagging, parallel remesh and construction of new sub-domain. Specifically, remeshing includes parallel remeshing of local grid and then updating the entire global grid (connectivity). Construction of new sub-domains includes re-ordering of cells and faces, calling ParMETIS for load balancing, defining the overlapping zone between partitions, data re-distribution, and defining new sub-domain data/variables. The strong scalability is investigated on these two primary procedures.

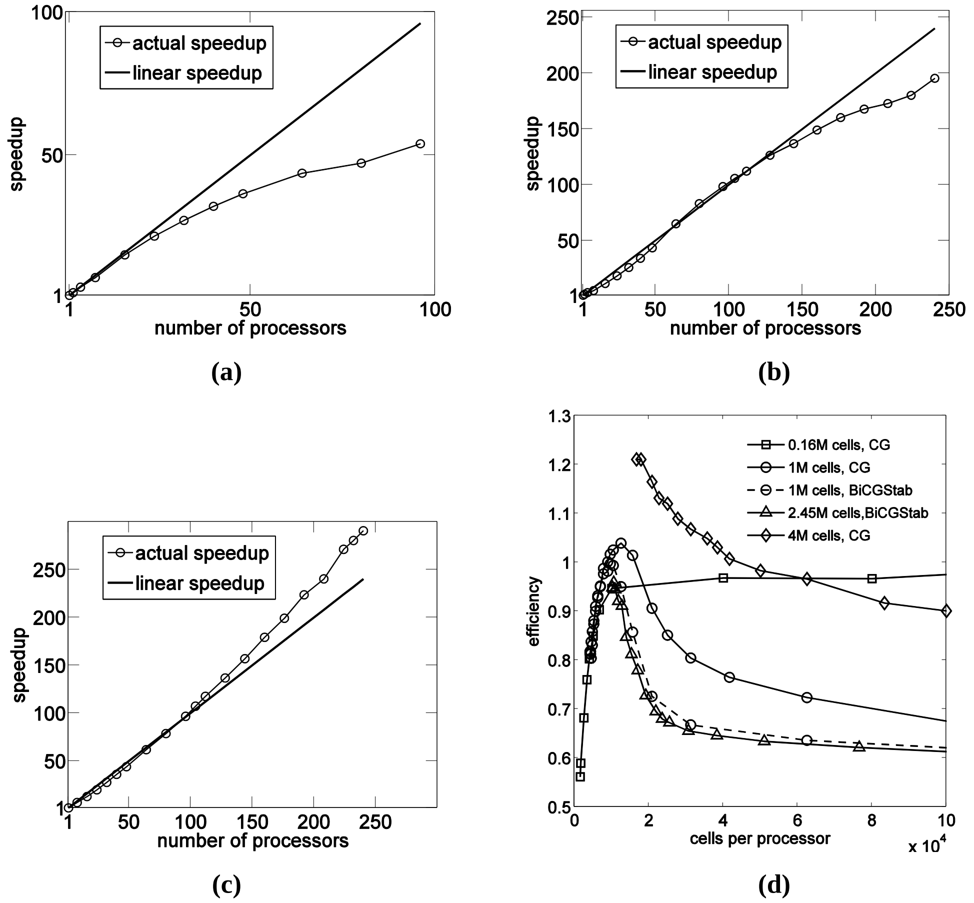


Figure 11. Speedup and efficiency of Cartesian grid solver. (a)-(c) Speedup as a function of number of processors for mesh size 1.6×10^5, 2×10^6 ,and 4×10^6. (d) Parallel efficiency with respect to number of cells per processor. The case marked with triangle is uniform flow past a circular cylinder with sharp interface method for solid interface. Since the cost of boundary condition is trivial compared with total cost of Cartesian grid solver, performance degradation due to imbalance of ghost cell distribution is not detected. Cartesian grid solver has best usage of computational power at 15,000 to 20,000 cells per processor. This range is determined by two factors: the computation to communication ratio and cache size of a machine.

Figure 12 presents overall strong scalability of remesh and sub-domain construction. These two sub-processes of AMR contribute comparable wall-clock time in all cases. Speedup of the AMR slows down around 64 processors and then level off at 128 processors. The cause of slow down on remeshing is mainly due to serial operation on updating connectivity of global grid after completing grid-up of each partition. Since possession of global grid in every processor is required for dynamically partitioning and determining communication pattern on unstructured grid, updating connectivity on global grid is unavoidable. Other overhead operation in AMR comes from all-to-all communication. All-to-all communication is used by the re-ordering cell in sub-domain construction. When it is utilized for scattering large data, it will cause significant deterioration to the performances.

We study weak scalability of AMR operation by fixing mesh size per processor to 6.4×10^4. Weak scalability can provide us useful understanding on the feature of current cell-based unstructured AMR. Especially, a program using intense far-end communication such as all-to-all communication should show unfavorable weak scalability. This type of communication is adopted in updating connectivity and reordering cell of entire domain. On the other hand, tasks involving nearest-neighbor communication are generally free from communication overhead thus good weak scalability is expected.

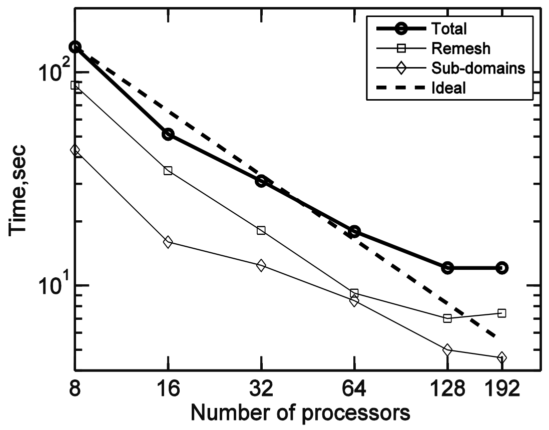
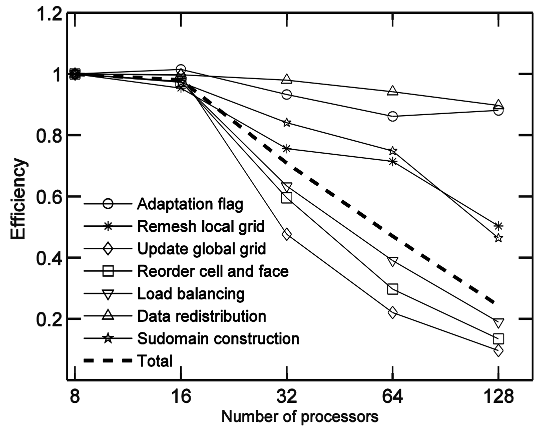


Figure 12. Wall-clock time of an AMR operation on strong scaling basis. Original mesh size is 8.19×10^6 and 2% of grid points are refined. Both two primary procedures of AMR, remesh and sub-domain construction, scale up to 128 processors with efficiency 0.68. All-to-all communication is the major cause of speedup slowdown in the remeshing and reordering procedures.

In this experiment, the number of processor involved spans from 8 to 128, which has total mesh size range from 5.12×10^5 to 8.192×10^6. When doubling number of processor, the computational grid length is double in one spatial direction. We utilize ParMETIS to initialize grid decomposition that results in a nearly identical mesh size per partition. Every processor assigns adaption flag randomly on 2% of cells in its own partition. Each experimental test is repeated five times and averaged wall-clock time is recorded. We calculate the time required to complete following procedures: adaptation flag assignment; remesh of local grid and update of global grid, which are two major components of remeshing; reordering cell and faces after repartition, load balancing, data redistribution, and constructing sub-domain variables. Figure 13 shows efficiency and percentage contribution of each procedure. The efficiency is calculated against results from 8 processors.

Overall efficiency of AMR are 0.99, 0.73, 0.48 and 0.26 at 16, 32, 64, 128 processor-count. However, when looking at the each procedure, adaptation flag and data re-distribution scales to 128 processors. These two procedures merely have communication between a partition and its neighbors. Efficiency of remeshing local gird and construction sub-domain variables reaches 0.5 at 128 processors. When number of processor increases, updating global grid and reorder cell and face are two most inefficiency procedures. Serial computation and communication overhead are primary sources of inefficiency in these two procedures. We use serial operations on some part of updating global grid, and all-to-all communication to updating global indices. Because of using all-to-all communication, we observed that the majority of communication overhead comes from the saturation of system send buffer. Due to the large amount of data and message counts, send calls waste time to wait for available system buffer. Even with the current non-blocking communication routines addressed, messages are delayed by system buffer. In some cases, we can alternatively circumvent the requirement of all-to-all communication by serial computation on global grid data in every processor, but this approach incurs serialization that hamper speedup. Using user-provided buffering MPI_Bsend may amortize this problem.



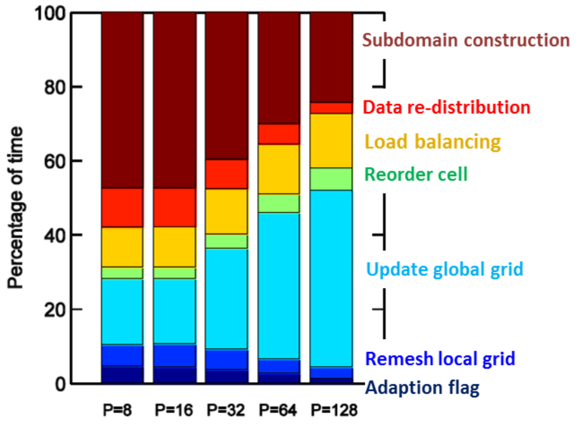
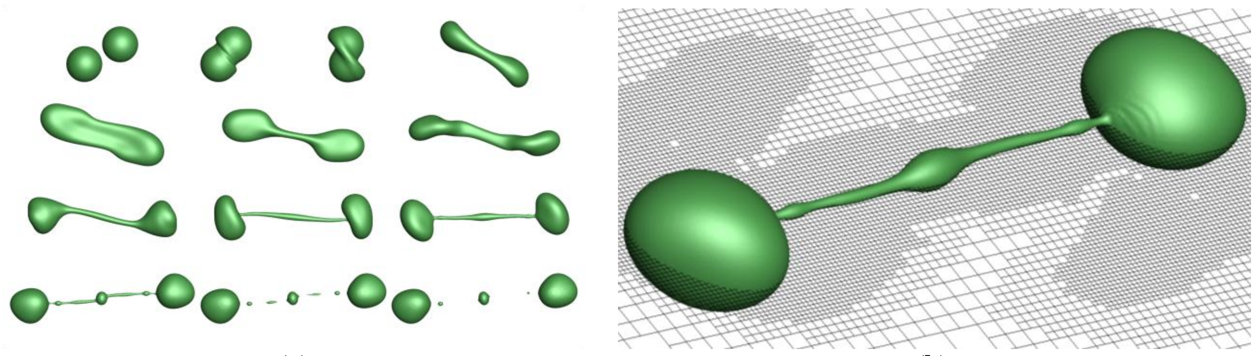


Figure 13. Efficiency and breakdown of an AMR operation on weak scaling basis. The mesh size per processor is 6.4×10^4 and 2% of grid points are refined. The total wall-clock time of one AMR operation is 2.79, 2.83, 3.85, 5.74, and 10.83 second for 8 to 128 processors. Procedures "Adaption flag", "remeshing local grid", "data redistribution", and "sub-domain construction" using the near-end communication pattern scale better than those counterpart procedures using all-to-all communication such as "updating global grid" and "reordering cell and face". Updating global grid become the major performance hurdle due to serialization and communication overhead.

C. Strong Scaling of a Practical Problem: Binary Droplet Collision

We describe a practical example, binary droplet collision to illustrate the overall performance of parallel adaptive Eulerian-Lagrangian method. This investigation concentrates on parallel performance of adaptive mesh refinement and field solver. Performance is highly dependable on the effectiveness of decomposition of Eulerian and Lagrangian domains and communication

In this experiment, we record wall-clock time of computation across 62 time-steps, with AMR operation applied 18 times. Such adaptation frequency is aggressive. AMR operation is activated by the requirement to refine Cartesian grid around moving Lagrangian interfaces every 4 time-steps, and solution-based adaption is considered in every 20 time-steps. For an ordinary problem, adaption can be less strict by enlarging the spatial range of refinement and decreasing AMR check frequency. We evaluate strong scalability on initial grid with 2.2×10^6 cells and six levels of refinement. The problem is run from 8 to 192 processors.



(a) **(b)**

Figure 14. Two droplets collide eccentrically to each other at We = 60.1 and Re = 302.8. It involves moving interface tracking, Lagrangian mesh modification (coarsen, smooth, and refine), interfaces reconstruction algorithm and adaptive mesh refinement techniques. This simulation takes about 4 days to complete a serial computation. With parallel implementation on 32 processors, it takes about 3.5 hours. (a) Morphological evolution during collision. (b) Snapshot of interface on adaptive Cartesian grid. This demonstration uses six level of refinement on initial mesh 2.2×10^6 cells.

Figure 14(a) and (b) shows history of droplets' morphological evolution and a snapshot of interfaces with sliced view of adaptive Cartesian grid. Weber number and Reynolds number are 60 and 302, and impact factor is 0.55. The definition of impact factor is the ratio of radial offset of two droplets center over the droplet diameter. The implemented adaptive mesh refinement algorithm effectively refines grid at the location of fluid interface and coarsen grid where grid resolution not demanded.

Figure 15 describe the executing time of primary procedures. We categorize these procedures according to their computation frame: interface shape modification (Lagrangian); surface tension computation and material determination (Eulerian-Lagrangian); projection of Cartesian grid solver (Eulerian); and AMR. Adaptive mesh refinement procedure shows least speedup and level off after 32 processors. The reason of low AMR efficiency is a result of load imbalance due to localized adaptation around interface. Frequent request of AMR also increase ratio of serial computation in the simulation. Computation time of AMR is from 4 to 13 % percent of the total wall-clock time. Even with high frequent calling of AMR algorithm, computational resources are mainly consumed by the Eulerian-Lagrangian flow solver. Note that among all tests, we observe that every processor has partial of Lagrangian interface, but loading of Lagrangian computation is not ideally balanced.

Pure Lagrangian procedures such as interface modification including smoothing, refining, and coarsening of Lagrangian interface occupies constant portion of total run time through all processors count, and its speedup levels off at 128 processors. The Eulerian-Lagrangian procedure shows lower parallel performace than Eulerian procedures, which is mainly due to the lower computation-to-communication ratio. Eulerian procedures scales well as we observed in stand-alone flow solver test. In summary, Parallel performance of each group are proportional to their computation-to-communication ratio, that is, in the descending rank as Eulerian, Eulerian-Lagrangian, Lagrangian and then AMR.

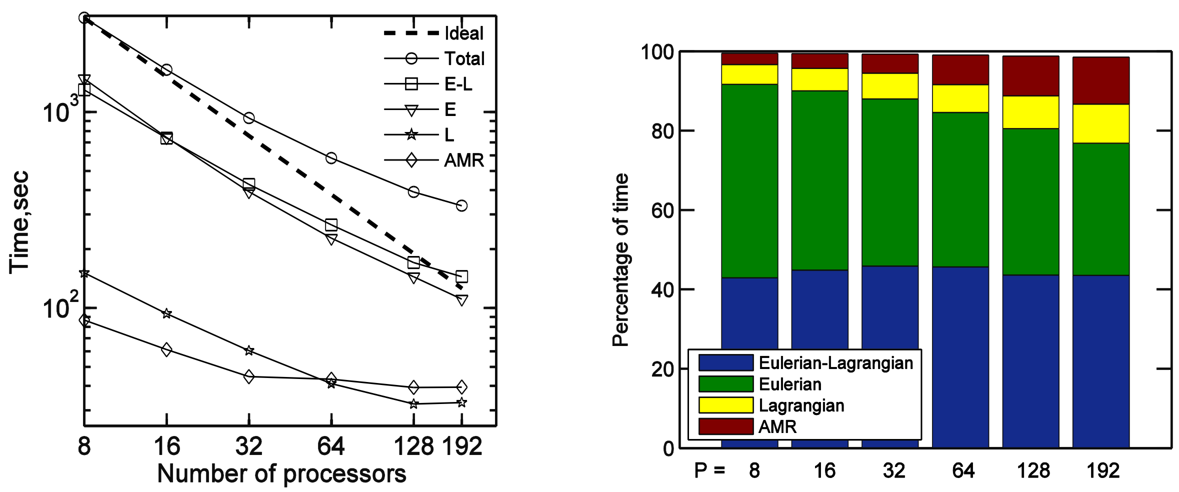


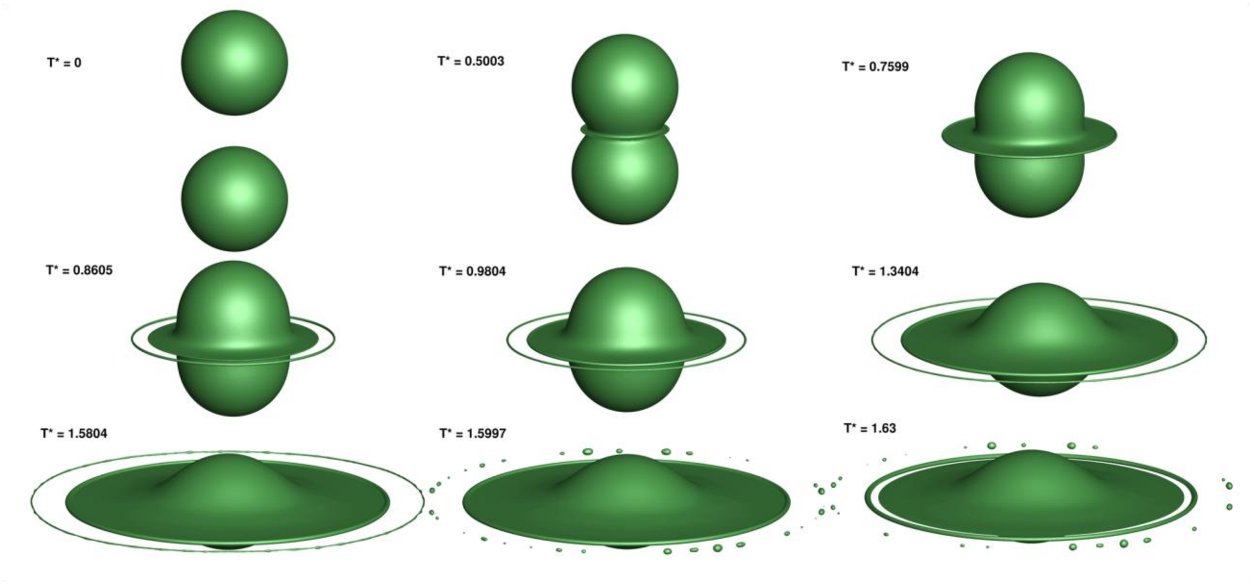
Figure 15. Executing time of the Eulerian-Lagrangian method for binary droplet collision computation. Original mesh size is 2.2×10^6. Time is recorded for 62 time-steps computation with AMR operation applied 18 times. Overall efficiency is 0.65 and 0.48 at 64 and 128 processors respectively. Procedures are categorized in four groups; Eulerian (E): operations of Cartesian grid solver; Eulerian-Lagrangian (E-L): cell material determination and surface tension computation; Lagrangian (L): interface shape modification; AMR: adaptive mesh refinement. Parallel performance of each group are proportional to their computation-to-communication ratio, that is, Eulerian > Eulerian-Lagrangian > Lagrangian > AMR. The load imbalance of Lagrangian data also incurs slowdown of speedup. Contribution of the wall-clock time from each group are stacked on the bar chart. Eulerian-Lagrangian computation occupies 40% to 45% of wall-clock time. With more processors, the ratio of wall-clock time spent on Lagrangian increases due to work imbalance.

We observed that surface tension computation costs 20 to 23 % run time. Besides, it is imbalance work among all processor that some of the processors fall behind to start the following procedure, Runge-Kutta/Crank-Nicolson solver (RK-CN). The consequence of "racing" of surface tension computation is a delay of communication in RK-CN solver. As a result, performance of RK-CN solver degrades. We observed RK-CN solver's trend on wall-clock time is similar to that of interface modification and surface tension computation, which is the impact of imbalance loading of Lagrangian computation.

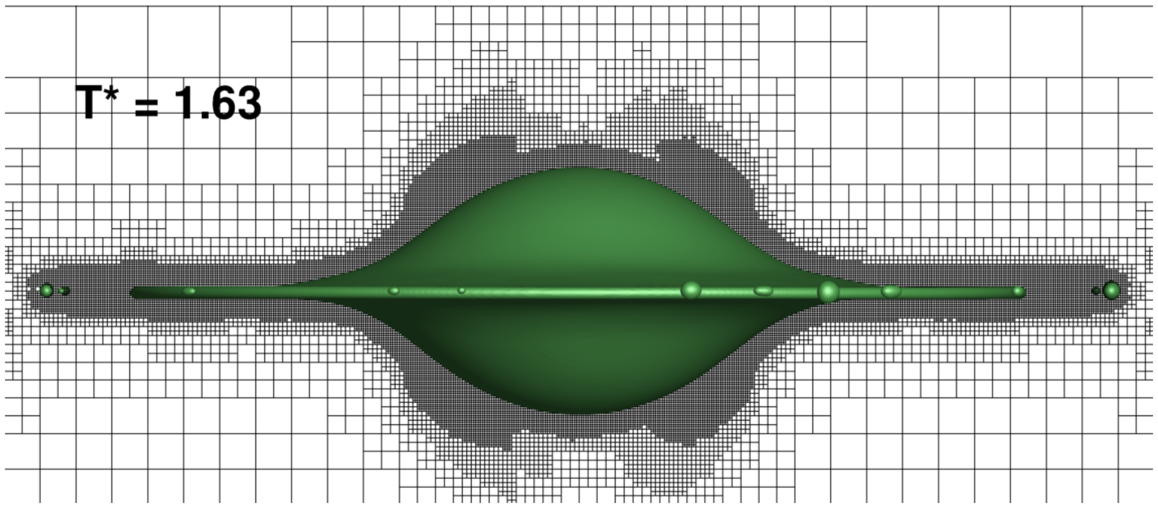
Overall, Eulerian-Lagrangian computation uses 40 to 45 percentage of wall-clock time. In addition, with more processors involving in computation, we observed more overhead of Lagrangian works. This effect induces more time spending on Lagrangian computation at higher processor count. These observations support our design of using marker locality to decompose Lagrangian domain. First, Eulerian-Lagrangian procedures are significant in terms of computation, that atomic decomposition will suffer communication overhead due to the frequent message and data size of communication. On the other hand, with task parallelism for Lagrangian domain, the intense computation work is not easy to balance, and most important, Lagrangian data has dependence on Eulerian data that task parallelism is difficult.

D. Binary Droplet Collision at High Weber Number

We further apply the present parallel adaptive Eulerian-Lagrangian method to simulate computationally demanding head-on binary droplet collision at high Weber number. By increasing the inertial force, the colliding droplets experience more violent deformation and breakup. The material used for two phases are water and air in room temperature at We = 1520 and Re = 8750. The mesh size is 2.4 million grid point initially with 6-level of refinement and reaches 7.2 million at the end of simulation. The finest grid resolution is 1/128 of initial droplet diameter. Figure 16(a) shows collision results obtained from current approach. The impinging flow from both sides of two droplets extrudes a circular disk at the waist of the merged body, which expands radially with a growing circular rim attached by the end-pinching mechanism due to capillary effect. The rim separates from the disk and continues to expand in radial direction because of inertia of itself. As the rim expands, the wavy structures are observed along the circumference of the rim, which is so called Rayleigh instability induced by capillary force. The rim eventually breaks into multiple secondary droplets as single water jet from a faucet breaks into a row of droplets. Current result qualitatively matches with the experiment pictures taken from[32]. Figure 16(b) presents the snapshot of the droplet with adaptive Cartesian mesh at the moment of secondary droplets formation. We will investigate the collision behavior and interfacial physics in the future.



(a)



(b)

Figure 16. Head-on binary droplet collision at We = 1520 and Re = 8750. (a) Morphological evolution during collision. (b) Snapshot of interface on adaptive Cartesian grid. The size of secondary droplets is about 100 times smaller than the original droplet size. Multiple-scale features of moving interface are successfully addressed with the present parallel adaptive Eulerian-Lagrangian method.

V. Conclusion

In this paper, we propose a parallel, adaptive Eulerian-Lagrangian method for large-scale moving boundary computation. The interactions between Eulerian and Lagrangian domain result in difficulties of decomposition and incur communication complexities between two individual domains. A spatial domain decomposition parallelism based on the locality of Lagrangian markers with respect to Eulerian partitions is adopted for reducing data size and frequency of communication. A parallel cell-based unstructured AMR algorithm is described with the analysis of details of AMR procedures including parallel grid generation of Eulerian sub-domains, global indices generation technique, domain re-decomposition and data redistribution. The highlights of parallel performance of the adaptive Eulerian-Lagrangian method on multiphase moving boundary problem are addressed as follow.

1) In the Eulerian domain decomposition including sharp interface, the current Cartesian grid solver achieves the most efficient computation in the range of 15,000 to 20,000 cells per processor. This range is determined by computation-to-communication ratio and the shared cache size per CPU.
2) Under strong scaling condition, the current cell-based unstructured AMR shows overall outstanding efficiency of 0.66 at 128 processors for a mesh size of 8.19×10^6 although the complexity and interactions between Eulerian and Lagrangian domain usually degrades the efficiency. Under weak scaling condition, the proposed cell-based unstructured AMR shows good scalability on procedures having near-end communication. However, the updating of global grid connectivity degrades the efficiency due to the use of all-to-all communication and serial computation.
3) Using parallel adaptive Eulerian-Lagrangian method, the simulation of binary droplet collision having satellite droplets shows scalable dynamic adaptive computation, and multi-scale features of interface are resolved. The overall efficiency is 0.65 at 64 processor for a 2.2-million mesh size problem. The overhead due to imbalanced Lagrangian work is the major factor of efficiency degradation at large number of processors.

Acknowledgement

The work reported in this paper has been partially supported by NASA Constellation University Institutes Program (CUIP), Claudia Meyer and Jeff Rybak program managers. We have benefited from the discussion with John Peugeot and Jeff West of NASA Marshall Space Flight Center. Special thanks go to Dr. Chang-Kwon Kang at University of Michigan for discussion regarding the parallel implementation and algorithms.

References

[1]Kuan, C.-K., Sim, J., and Shyy, W., "Adaptive thermo-fluid moving boundary computations for interfacial dynamics," *Acta Mechanica Sinica*, Vol. 28, No. 4, 2012, pp. 999-1021.

[2]Sim, J., Kuan, C.-K., and Shyy, W., "Simulation of Spacecraft Fuel Tank Self-Pressurization Using Eulerian-Lagrangian Method," *49th AIAA Aerospace Science Meeting Including the New Horizons Forum and Aerospace Exposition*, AIAA, Washington, DC, AIAA-2011-1318, 2011.

[3]Shyy, W., Correa, S. M., and Braaten, M. E., "Computation of Flow in a Gas Turbine Combustor," *Combustion Science and Technology*, Vol. 58, No. 1-3, 1988, pp. 97-117.

[4]Singh, R., and Shyy, W., "Three-dimensional adaptive Cartesian grid method with conservative interface restructuring and reconstruction," *Journal of Computational Physics*, Vol. 224, No. 1, 2007, pp. 150-167.

[5]Sim, J., and Shyy, W., "Interfacial flow computations using adaptive Eulerian-Lagrangian method for spacecraft applications," *International Journal for Numerical Methods in Fluids*, Vol. 68, No. 11, 2012, pp. 1438-1456.

[6]Uzgoren, E., Sim, J., and Shyy, W., "Marker-based, 3-D adaptive Cartesian grid method for multiphase flow around irregular geometries," *Communications in Computational Physics*, Vol. 5, No. 1, 2009, pp. 1-41.

[7]Kuan, C.-K., Sim, J., and Shyy, W., "Parallel, Adaptive Grid Computing of Multiphase Flows in Spacecraft Fuel Tanks," *50th AIAA Aerospace Science Meeting Including the New Horizons Forum and Aerospace Exposition*, AIAA, Washington, DC, AIAA-2012-761, 2012.

[8]Tryggvason, G., Bunner, B., Esmaeeli, A., Al-Rawahi, N., Tauber, W., Han, J., Jan, Y. J., Juric, D., and Nas, S., "A front-tracking method for the computations of multiphase flow," *Journal of Computational Physics*, Vol. 169, No. 2, 2001, pp. 708-759.

[9]Uzgoren, E., Singh, R., Sim, J., and Shyy, W., "Computational modeling for multiphase flows with spacecraft application," *Progress in Aerospace Sciences*, Vol. 43, No. 4-6, 2007, pp. 138-192.

[10]Espostiongaro, T., Cavazzoni, C., Erbacci, G., Neri, A., and Salvetti, M., "A parallel multiphase flow code for the 3D simulation of explosive volcanic eruptions," *Parallel Computing*, Vol. 33, No. 7-8, 2007, pp. 541-560.

[11]Sussman, M., "A parallelized, adaptive algorithm for multiphase flows in general geometries," *Computers & Structures*, Vol. 83, No. 6-7, 2005, pp. 435-444.

[12]Marella, S. V., "A Parallelized sharp-interface fixed grid method for moving boundary problems," Ph.D. Dissertation, Mechanical Engineering Dept., University of Iowa, Iowa City, IA, 2006.

[13]Agbaglah, G., Delaux, S., Fuster, D., Hoepffner, J., Josserand, C., Popinet, S., Ray, P., Scardovelli, R., and Zaleski, S., "Parallel simulation of multiphase flows using octree adaptivity and the volume-of-fluid method," *Comptes Rendus Mecanique*, Vol. 339, No. 2-3, 2011, pp. 194-207.

[14]Darmana, D., Deen, N. G., and Kuipers, J. A. M., "Parallelization of an Euler-Lagrange model using mixed domain decomposition and a mirror domain technique: Application to dispersed gas-liquid two-phase flow," *Journal of Computational Physics*, Vol. 220, No. 1, 2006, pp. 216-248.

[15]Agbaglah, G., Delaux, S. e. b., Fuster, D., Hoepffner, J. e. r., o, m., Josserand, C., Popinet, S. e. p., Ray, P., Scardovelli, R., and Zaleski, S. e. p., "Parallel simulation of multiphase flows using octree adaptivity and the volume-of-fluid method," *Comptes Rendus Mecanique*, Vol. 339, No. 2-3, pp. 194-207.

[16]Burstedde, C., Wilcox, L. C., and Ghattas, O., "Scalable algorithms for parallel adaptive mesh refinement on forests of octrees," *SIAM Journal on Scientific Computing*, Vol. 33, No. 3, 2011, pp. 1103-1133.

[17]Berger, M., and Oliger, J., "Adaptive mesh refinement for hyperbolic partial differential equations," *Journal of Computational Physics*, Vol. 53, No. 3, 1984, pp. 484-512.

[18]Gunney, B. T. N., Wissink, A. M., and Hysom, D. A., "Parallel clustering algorithms for structured AMR," *Journal of Parallel and Distributed Computing*, Vol. 66, No. 11, 2006, pp. 1419-1430.

[19]MacNeice, P., Olson, K. M., Mobarry, C., de Fainchtein, R., and Packer, C., "PARAMESH: A parallel adaptive mesh refinement community toolkit," *Computer Physics Communications*, Vol. 126, No. 3, 2000, pp. 330-354.

[20]Deiterding, R., "A parallel adaptive method for simulating shock-induced combustion with detailed chemical kinetics in complex domains," *Computers Structures*, Vol. 87, No. 11-12, 2009, pp. 769-783.

[21]Powell, K. G., Zeeuw, D. L. D., Sokolov, I. V., Tamas, I., and Stout, Q., "Parallel, AMR MHD for Global Space Weather Simulations," *Adaptive Mesh Refinement - Theory and Applications*. edited by Plewa, T., Linde, T., and Weirs, V. G., Vol. 41, Springer, Berlin Heidelberg, 2005, pp. 473-490.

[22]Griffith, B. E., Hornung, R. D., McQueen, D. M., and Peskin, C. S., "Parallel and Adaptive Simulation of Cardiac Fluid Dynamics," *Advanced Computational Infrastructures for Parallel and Distributed Adaptive Applications*. 1st ed., edited by Parashar, M., and Li, X., John Wiley and Sons, 2009, pp. 105-130.

[23]Zuzio, D., and Estivalezes, J. L., "An efficient block parallel AMR method for two phase interfacial flow simulations," *Computers & Fluids*, Vol. 44, No. 1, 2011, pp. 339-357.

[24]Kirk, B. S., Peterson, J. W., Stogner, R. H., and Carey, G. F., "libMesh: a C++ library for parallel adaptive mesh refinement/coarsening simulations," *Engineering with Computers*, Vol. 22, No. 3-4, 2006, pp. 237-254.

[25]Lawlor, O. S., Chakravorty, S., Wilmarth, T. L., Choudhury, N., Dooley, I., Zheng, G., and Kale, L. V., "ParFUM: a parallel framework for unstructured meshes for scalable dynamic physics applications," *Engineering with Computers*, Vol. 22, No. 3-4, 2006, pp. 215-235.

[26]Balay, S., Brown, J., Buschelman, K., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Smith, B. F., and Zhang, H., "\url{http://www.mcs.anl.gov/petsc}."

[27]Falgout, R. D., and Yang, U. M., "hypre: a Library of High Performance Preconditioners," *Computational Science - ICCS 2002*. edited by Sloot, P. M. A., Tan, C. J. K., Dongarra, J. J., and Hoekstra, A. G., Vol. 2331, Springer Berlin, Heidelberg, 2002, pp. 632-641.

[28]Schamberger, S., and Wierum, J.-M., "Graph Partitioning in Scientific Simulations: Multilevel Schemes versus Space-Filling Curves," *Parallel Computing Technologies*. edited by Malyshkin, V., Vol. 2763, Springer Berlin, Heidelberg, 2003, pp. 165-179.

[29]Plimpton, S., "Fast parallel algorithms for short-range molecular dynamics," *Journal of Computational Physics*, Vol. 117, 1995, pp. 1-19.

[30]Lhner, R., "Parallel unstructured grid generation," *Computer Methods in Applied Mechanics and Engineering*, Vol. 95, 1992, pp. 343-357.

[31]Chrisochoides, N., "Parallel Mesh Generation," *Numerical Solution of Partial Differential Equations on Parallel Computers*. 1st ed., edited by Bruaset, A. M., and Tveito, A., Vol. 51, Springer-Verlag, New York, 2006, pp. 237-259.

[32]Pan, K.-L., Chou, P.-C., and Tseng, Y.-J., "Binary droplet collision at high Weber number," *Physical Review E*, Vol. 80, No. 3, 2009, 036301.