Hindawi
Scientific Programming
Volume 2018, Article ID 6387049, 14 pages
https://doi.org/10.1155/2018/6387049

Research Article

Load Balancing for Parallel Multiphase Flow Simulation

Najeeb Ahmad, Muhammad Nufail Farooqi, and Didem Unat

Koç University, Sarıyer, 34450 Istanbul, Turkey

Correspondence should be addressed to Najeeb Ahmad; nahmad16@ku.edu.tr

Received 5 September 2017; Accepted 30 January 2018; Published 7 March 2018

Academic Editor: Marco Aldinucci

Copyright © 2018 Najeeb Ahmad et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

This paper presents a scalable dynamic load balancing scheme for a parallel front-tracking method based multiphase flow simulation. In this simulation employing both Lagrangian and Eulerian grids, processes operating on Lagrangian grid are susceptible to load imbalance due to moving Lagrangian grid points (bubbles) and load distribution based on spatial location of bubbles. To load balance these processes, we distribute load keeping in view both current processor load distribution and bubble spatial locality and remap interprocess communication. The result is a uniform processor load distribution and predictable and less expensive communication scheme. Scalability studies on the Hazel Hen supercomputer demonstrate excellent scaling with exponential savings in execution time as the problem size becomes increasingly large. While moderate speedup is observed for strong scaling, speedup of up to 30% is achieved over nonload-balanced version when simulating 13824 bubbles on 4096 cores for weak scaling studies.

1. Introduction

Multiphase flow in fluid mechanics refers to the simultaneous flow of a mixture of materials with different chemical properties or different phases (e.g., solid, liquid, or gas). Phenomena such as rainfall, floods, oil, and gas flow in pipelines, fog and mist, water and steam mixture in boilers and condensers, the flow of toner in printers, and blood flow in blood vessels are examples of multiphase flow to name a few. Given the widespread prevalence of such processes, it is important to study and accurately predict flow behavior of these processes to better understand and, in the case of industrial processes, to improve their efficiency and effectiveness [1]. Because of the practical difficulties of performing experimental studies on multiphase flows, computer simulation has become an essential tool that can yield great insights into fluid flow behaviors [2]. Various computational methods for multiphase flow have been presented in literature including volume of fluid [3, 4], phase field [5, 6], level set [7, 8] and front-tracking method [9, 10]. These techniques mainly differ in the treatment of boundary among fluids termed as *interface* or *front*. The first three approaches discussed above are based on front capturing method where the interface between the fluids is implicitly represented as part of the structured grid (Eulerian-Eulerian method) while the front-tracking approach uses a separate unstructured grid for interface representation and tracking (Eulerian-Lagrangian method).

For the purpose of accuracy and stability, both front capturing and front-tracking methods require high spatial and temporal resolution. From the perspective of computer simulation, this means a higher computational requirement and, in terms of computation time, a serial implementation may not be practical in most cases. Therefore, most computational simulations of these methods are implemented in parallel. One of the parallel implementations of multiphase flow based on front-tracking method is presented in [11]. Authors parallelize both Eulerian and Lagrangian grid computations with Lagrangian grid computations performed by a separate group of processes. In their strategy, processes operating on the Eulerian grid are termed as *subdomain* processes while ones operating on the Lagrangian grid are called *subfront* processes. While equally divided Eulerian grid parts are distributed among subdomains, the distribution of interfaces or *bubbles* among subfront processes is based on spatial location of bubbles in the Eulerian grid. Depending upon the initial bubble placement and bubble movement during the course of the simulation, each subfront process may not get the same number of bubbles creating a load imbalance among subfront

processes. In their parallelization strategy, a subfront process communicates with a predetermined fixed number of subdomains. We will refer to this strategy as *fixed mapping* strategy in the subsequent discussions. The fixed mapping strategy requires bubble data sharing among subfront processes for the bubbles crossing the process boundaries. This introduces additional communication among subfront processes and makes an exact analysis of subfront-subfront communication difficult.

This work is built upon the implementation discussed in [11] with an emphasis on load balancing and devises a more predictable communication scheme for subfront processes. We distribute work among subfront processes based on the output of a load balancing algorithm [12] and eliminate bubble sharing among subfront processes. Contributions of this work can be summarized as follows:

(i) We perform load balancing on subfront processes by treating interface or bubble distribution problem (a set of Lagrangian grid points) as a particle distribution problem (single point) for subfront processes.

(ii) We improve interprocess communication scheme to incorporate *adaptive subdomain-subfront mapping* replacing fixed mapping, which results in more predictable and less expensive subfront-subfront communication.

(iii) We perform strong and weak scaling studies in a realistic setting. For strong scaling, we obtain a 5% speedup for adaptive over fixed mapping scheme simulating 864 bubbles on 512 cores and for weak scaling; we achieve up to 30% speedup over fixed mapping simulating 13824 bubbles on 4096 cores.

In the rest of the paper, after discussing related work in Section 2, we give an overview of communication overheads, load distribution, and load imbalance for the fixed subdomain-subfront mapped simulation in Section 3. In Section 4, we present our load balancing and the adaptive subdomain-subfront mapping communication strategy followed by a discussion on our strong and weak scaling results in Section 5. Finally, we present our conclusions in Section 6.

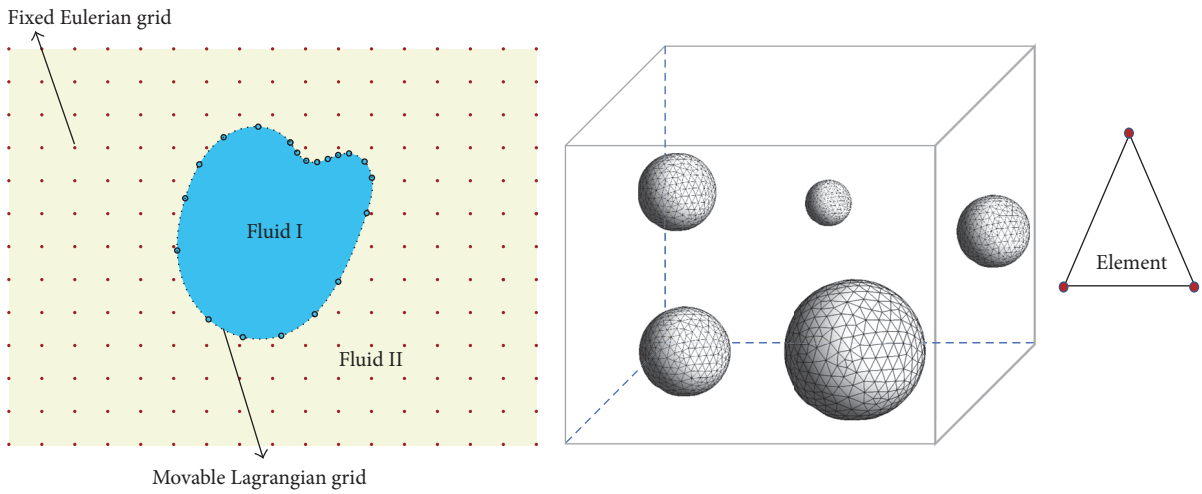
2. Related Work

Applications that have an evolving geometry and movable meshes are susceptible to load imbalance because of their dynamic structure [13]. For multiphase flow simulations, a number of load balancing strategies are proposed in the literature. Watts et al. [14] presented the first effort for dynamic load balancing of multiphase computations. Their strategy is based on general load balancing model proposed in [15] suitable for single-phase flow simulations. To make it suitable for multiphase flows, they introduce the concept of vector view of the load. In this methodology, load for each phase is treated separately and load imbalance calculated using modified load balance equations based on load per phase. The rest of load balancing steps are treated same as in [15] except for replacing scalar load quantities with vector ones. They tested their technique on the Hawk particle simulator [16] and Concurrent Graph Library for particle simulations. In both cases, better application efficiency was achieved with vector load balancing over the scalar load balancing.

In the Large Eddy Simulation of multiphase flows for gas turbines, Ham et al. [17] employed ParMETIS [18] for dynamic load balancing by stopping the simulation at a load imbalance threshold, performing load balancing, and restarting the application. They argue that simulation stop/start strategy is feasible because of low overhead of restart time. They tested their strategy for a small model particle cost distribution only and have shown to achieve a theoretical speed up of 4x over nonload-balanced version. They, however, admit that the multiconstraint options used in ParMETIS partitioning increase edge cuts and may reduce performance in real applications.

In [19], Borrell et al. presented their parallel implementation of the volume of fluid (VOF) method with dynamic load balancing. In VOF, load imbalance occurs due to the nonhomogeneous distribution of interface in the domain. They devise a five-step procedure for load balancing constituting load imbalance determination, interprocess communication scheme determination, load distribution to under loaded tasks, computation, and finally collection of results by the parent tasks. Their load balancing version is shown to outperform the standard version by a speedup of 3x. Another VOF based parallel simulation by Agbaglah et al. [20] uses octrees to refine grid at point of interests and devise a strategy to handle processor load imbalance resulting due to grid adaptivity. They argue that good initial partitioning can be achieved in practice using a heuristic bubble partitioning algorithm. Having obtained good initial load partitioning, they use weighted graphs to move load among processors to keep load balanced and to minimize interprocessor communication. Both of these implementations deal with single type of mesh. Arguing that VOF method is not well suited for implementation on modern GPU accelerators, Ikebata and Xiao [21] devised a new multiphase flow computational model that is easily portable and reasonably scalable on modern GPUs. Their method is based on front capturing method and the implementation employs nonconformal domain decomposition to improve load balancing and computation time. For data parallel implementations of simulations like multiphase flow, Aldinucci et al. [22, 23] presented a framework for simplifying the implementation and increasing portability and scalability of such applications on heterogeneous platforms (CPUs + GPUs).

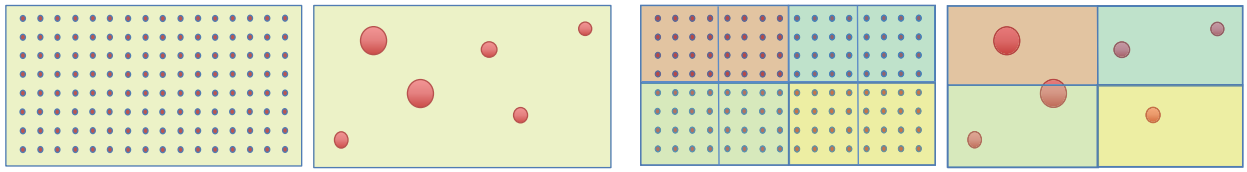
In [24], Houzeaux et al. proposed dynamic load balancing for particle flow through fluids using dynamic load balancing library [25]. The library requires the use of hybrid programming model (MPI + OpenMP). Load balancing is achieved by relinquishing resources of a blocked MPI process and using them instead for spawning more OpenMP threads for another process on the same node increasing application parallelism. They achieved a speedup of up to 3.5x for 10 M particles simulation and up to 1.5x for 0.5 M particle simulation. This technique, however, has the limitation of using hybrid programming model and of providing implicit load balancing for processes residing on the same node only.



(a) 2D representation of 3D Eulerian and Lagrangian grids

(b) Multiple Lagrangian grid interfaces in 3D domain. Lagrangian grid is comprised of triangular elements as depicted

Figure 1: Eulerian and Lagrangian grids.



(a) Separation of Eulerian (left) and Lagrangian (right) grids into separate computational units

(b) Subdivision of domain and front into subdomains and subfronts. Similarly colored subfronts and subdomains are mapped to each other

Figure 2: Parallelism in fixed mapping front-tracking multiphase flow simulation.

The previous work on load balancing multiphase flow simulations has mostly focused on either load balancing particles or front capturing methods employing single grid for the treatment of interface. In this work, we focus on front-tracking method employing two grids for treating fluid and interface separately (Eulerian-Lagrangian) and balance processes for the Lagrangian grid. We develop necessary data structures and communication strategy for dynamic load balancing of Lagrangian grid processes.

3. Multiphase Flow Simulation

Before discussing contributions of this study, it is pertinent to give an overview of the existing front-tracking simulation based on [11]. This simulation is a parallel implementation of front-tracking method developed by Unverdi and Tryggvason [26]. The method solves governing flow equations for the entire domain by appropriately taking into account material properties like density and viscosity and properly treating interfacial surface tension. Both Eulerian and Lagrangian grids are employed for this purpose. While flow equations are solved on the stationary Eulerian grid, boundary or interface among different materials/fluids is tracked with a movable Lagrangian grid. A 2D depiction of a rectangular Eulerian grid and Lagrangian grid is shown in Figure 1(a). The Lagrangian grid for interface representation uses triangular mesh with each element represented by three vertices as depicted in Figure 1(b).

The parallel simulation logically incorporates three levels of parallelism. At the first level, computations of the Eulerian grid (referred to as *Domain*) and Lagrangian grids (referred to as *Front*) are separated into distinct computational units. At the second level, both domain and front are further subdivided into *subdomains* and *subfronts* with each subdomain and subfront handled by a separate MPI process. At the third level, each process employs OpenMP multithreading wherever possible. The first two levels of parallelism are depicted in Figure 2. This separation and subdivision into processes, however, necessitates interprocess communication due to process interdependencies. A brief overview of interprocess communication is presented in the next subsection. A comprehensive explanation of interprocess communication can be found in [11].

3.1. Interprocess Communication. Three types of communication are introduced as a result of parallelization, namely, (1) *subdomain-subdomain*, (2) *subdomain-subfront*, and (3) *subfront-subfront*. Most computations on the Eulerian grid

are stencil computations requiring exchange of ghost cell data among neighboring subdomain processes, thus resulting in subdomain-subdomain communication. This communication has the highest cost compared with the other two. However, as the number of ghost cells exchanged in each iteration is fixed, the communication cost is fixed.

To update the location of *interfaces* in the simulation domain, subfront processes need to send interface data to subdomain processes and collect updated data back resulting in subdomain-subfront communication. Each subfront is mapped to one or more subdomain processes based on input process configuration and spatial location of subdomains. This mapping is done based on input file provided to the simulation by the user. Once initially done, the subdomain(s)-subfront mapping remains fixed throughout the course of the simulation. Subsequently, the subdomains exchange data with their mapped subfront only. One of the many possible mapping configurations is depicted in Figure 2, where a subfront is mapped to two subdomains. In terms of communication cost volume, this type of communication ranks as the second highest. The amount of data exchanged between subfront and its mapped subdomains depends upon the number of interfaces or bubbles contained by the subfront. As the number of interfaces within subfront may vary during the course of the simulation, this type of communication is variable.

Due to parallelization of the front, computations on bubbles are divided among subfront processes. This division of work depends upon the spatial location of bubbles in the Eulerian grid and may result in a bubble shared by multiple subfront processes. A bubble may be shared between as few as two and as many as eight subfronts. Once computations are performed on a bubble, bubble data needs to be sent to the appropriate subdomain(s) for position update. However, due to fixed subdomain-subfront mapping, owner subfront cannot directly send shared bubble data to the subdomain(s) other than the ones it is mapped to. Moreover, as the bubbles move in the spatial domain during the course of the simulation, a bubble's center may fall into the purview of a different subfront. This bubble movement from one subfront to another initiates ownership transfer from the previous owner to the new one. In ownership transfer, all bubble data is sent from the previous owner to the new owner subfront resulting in another type of subfront-subfront communication.

The number of subfronts sharing bubbles and number of bubbles requiring ownership transfer may vary during the course of the simulation, which makes it difficult to approximate subfront-subfront communication. One of the contributions of this study is to make subfront-subfront communication more predictable.

3.2. Load Distribution. Subdomain processes are mainly responsible for two types of computations. Firstly, they solve flow equations on the Eulerian grid and secondly they update the spatial position of bubble points lying in their jurisdiction as received from subfront processes. Owing to the domain parallelization strategy, distribution of Eulerian grid points

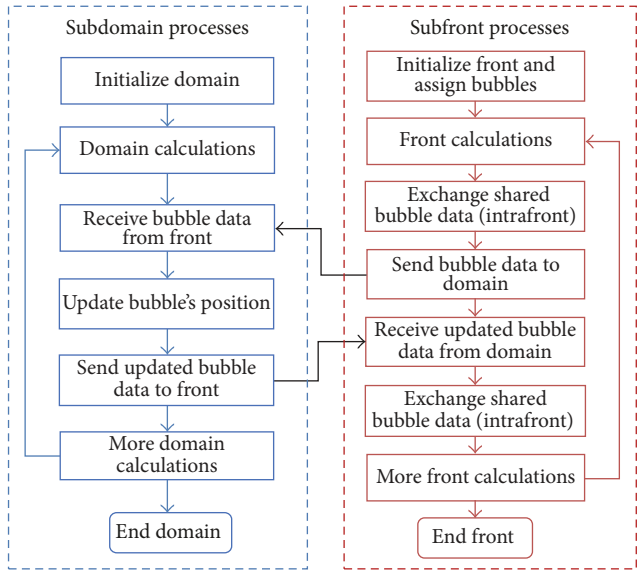


Figure 3: Simplified flow chart of fixed mapping parallel multiphase flow simulation. Black arrows indicate interprocess communication.

across subdomain processes is uniform. This implies that Eulerian grid computations across subdomain processes are fixed and, therefore, these processes are load-balanced with respect to Eulerian grid computations. The number of bubble points requiring position update is dependent upon spatial bubble distribution in the domain. This means that subdomain computational load for Lagrangian grid position update is variable across subdomains during the course of simulation.

Subfront processes are responsible for computations on the Lagrangian grid (e.g., calculating interface surface tension and interface surface normal). In the fixed subdomain-subfront mapping strategy, because the bubble point assignment to subfronts is based on spatial location of bubbles, subfront processes may get variable computational load resulting in load imbalance. This imbalance may get severe in the case of a large number of bubbles accumulating in subdomains mapped to only a few number of subfronts. The load balancing strategy devised in this work mainly addresses this type of imbalance as discussed in Section 4.

3.3. Program Structure. Figure 3 shows the simplified flow chart of subdomain and subfront processes, their intercommunication (subdomain-subfront) and intracommunication (subdomain-subdomain, subfront-subfront). The two sets of processes operate in parallel and synchronize at the end of each communication type. Each process starts with process initialization and performs computations on its respective Eulerian or Lagrangian grid. After initial bubble assignment, a subfront process sends bubble data to its mapped domains by exchanging shared bubble data among sharer subfronts (intrafront communication) and forms a single data packet for each mapped domain. On the other hand, subdomain

processes perform ghost cell exchange for stencil computations. This is followed by subdomain processes receiving bubble data from mapped subfront, updating bubble position and other parameters, and sending updated data back to respective subfronts. Finally updated bubble data is exchanged among sharer subfronts and then two sets of processes continue with their respective grid computations. This process repeats until a specified number of iterations have been achieved. All the communications are nonblocking employing communication and computation overlap as much as possible.

4. Load Balancing

For parallel computing systems, load balancing is an important consideration for efficient resource utilization and performance. Depending upon the type of parallel application, various load balancing strategies are available in literature. For mesh-based applications like multiphase flow simulation, graph-based and geometric partitioners are common options. These partitioners can be used for both static load balancing in which average system behavior is used to make load balancing decision or dynamic load balancing in which current system state dictates load balancing output. Geometric methods partition data based on the spatial locality of data objects. These methods use physical coordinates and weights of the objects to make partitioning decisions. Graph-based methods represent the application as a graph with data objects as vertices and their dependencies as edges of that graph. Partitioning is then performed keeping in view equal weight for the partitioned parts. When comparing graph-based versus geometric techniques, graph-based techniques are known to provide good results for mesh-based applications [27] at the cost of higher computation time and added implementation complexity. On the other hand, geometric partitioning methods are faster and easier to implement but may result in a higher interprocess communication. Applications not easily expressible as connected graphs are good candidates for geometric partitioning [27]. Particle simulation is one such example where geometric methods are readily applicable. Apart from these traditional approaches, some new methods based on modifications on traditional approaches like hybrid geometric/graph partitioners or hypergraph partitioners are also available in literature.

For the multiphase flow simulation under consideration in this study, the main source of load imbalance is the distribution of work based on spatial location of bubbles in the domain. Although both the subfront and subdomain processes suffer from this load imbalance, this work focuses on load balancing subfront processes and mapping resulting communication accordingly. The subfront processes operate on a Lagrangian grid consisting of distinct sets of interconnected grid points. One such set of points constitutes one interface or bubble and is chosen as one compute unit in our load balancing approach. For devising load balancing strategy for subfront processes, following objectives have been taken into account:

(i) Equally distribute bubbles among subfront processes.

(ii) Minimize number of subdomains that a subfront communicates with in order to reduce separate MPI send/receive posts for subfront-subdomain communication. This will help reduce overhead associated with posting MPI sends/receives.

(iii) Minimize subfront-subfront communication.

Keeping these requirements in view and by treating a single bubble or interface as a particle in particle simulation, geometric partitioning strategy seems a natural fit for balancing subfront processes. Geometric partitioning complies with our partitioning objectives by allowing equal bubble distribution based on their spatial location and minimizing distinct subdomain-subfront mapping by placing closely located bubbles together. When closely located bubbles are assigned to a single subfront process, it increases the probability of them lying in the same subdomain, thus reducing the number of distinct subdomain-subfront communications. As discussed earlier, the bulk of subfront-subfront communication mainly exists due to shared bubbles among subfront processes. Irrespective of the partitioning strategy, the requirement of reducing subfront-subfront communication can, therefore, be met by eliminating shared bubbles.

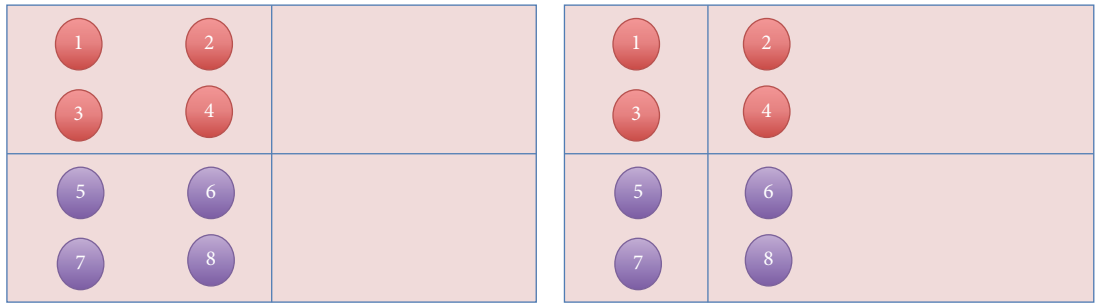
For the purpose of this study, we leverage one of the available load balancing tools. Several load balancing tools supporting geometric partitioning are available. DRAMA [28, 29] is a software library written for finite element methods that supports both graph and geometric partitioning methods. PLUM [30] is a framework for load balancing adaptive unstructured meshes supporting many partitioning algorithms. Zoltan [31] is a suite of tools from Sandia National Laboratories for dynamic load balancing and data migration. It supports a variety of traditional load balancing algorithms like geometric and graph-based as well as sophisticated nontraditional approaches like hypergraph partitioning algorithm [32]. Given application design flexibility, we decided to use Zoltan for our application.

The load balancing procedure for subfront processes can be described as a two-step process: (1) distributing bubbles among subfront processes based on one of the partitioning methods supported by Zoltan; (2) devising a new subfront-subdomain communication strategy due to revised subfront load assignment.

4.1. Distributing Bubbles. As described previously, geometric methods are a good choice for partitioning bubbles among subfront processes. Having decided the partitioning algorithm, bubble distribution among subfronts is performed as follows:

(i) *Initial bubble distribution*: before applying the partitioning process, bubbles are initially distributed among subfront processes as per previous strategy, that is, assigning bubbles to subfront processes based on their spatial location in the domain. This initial distribution is then used by the Zoltan library to make partitioning decision described next.

(ii) *Partitioning bubbles using Zoltan*: the first step towards partitioning using Zoltan is to write callback



(a) Bubble distribution before load balancing. 8 bubbles distributed between two subfront processes

(b) Bubble distribution after load balancing. Bubbles equally distributed among the four subfront processes

FIGURE 4: Sample bubble distribution among four subfront processes before and after load balancing.

functions providing information about the application to Zoltan. For geometric partitioning methods, depending on the algorithm used, at least four callback functions are needed. For Recursive Coordinate Bisection (RCB) method [12] used in this work, Zoltan requires the following types of callback functions to be provided by the programmer:

(a) `ZOLTAN_NUM_GEOM_FN` returns the number of items required to express geometry of the object. In our case, each bubble point is expressed as (x, y, z) coordinates. Hence, this function returns 3.

(b) `ZOLTAN_GEOM_FN` returns geometry values for the object under consideration. In our case, this function returns coordinates of the bubble center.

(c) `ZOLTAN_NUM_OBJ_FN` returns the number of objects currently assigned to the processor. In our case, it returns number of bubbles assigned to a subfront processor (as per initial distribution).

(d) `ZOLTAN_OBJ_LIST_FN` returns identifiers (IDs) of objects assigned to a processor. It returns local and global IDs of the assigned objects. Global IDs are used for object identification within Zoltan and are unique among all processes. Local IDs are not required by Zoltan and can be used by application as per their convenience. In our case, global and local IDs are assigned during input file read and subfront process initialization, respectively.

Once callback functions are in place, the Zoltan library is initialized using the `Zoltan_Initialize()` function. Then `Zoltan_Create()` is called with the MPI communicator of subfront processes to initialize memory for Zoltan operations. This is followed by setting partitioning method and mapping callback functions using the `Zoltan_Set_Param()` function. Finally, `Zoltan_LB_Partition()` is called to perform balanced bubble distribution over subfront processes. The output parameters of this function are then used to redistribute bubbles as described next.

(iii) *Initial bubble redistribution*: for each subfront process, the `Zoltan_LB_Partition()` function outputs parameters like the number of bubbles to be imported into and exported from the process, their local and global IDs, and IDs of processes bubbles to be imported from or exported to. Using this information, we redistribute bubbles among subfront processes. To avoid MPI communication, instead of transferring bubbles between processes, they are simply deallocated from previous owner processes and allocated to the new owners. This is possible because the initial redistribution takes place at the beginning of the simulation. A sample bubble redistribution scenario is shown in Figure 4.

(iv) *Dynamic bubble redistribution*: as the bubbles are movable in the domain, a new geometric partition may be required after certain number of iterations. We check whether the load is imbalanced after every specified number of iterations (provided as an input parameter to the simulation) using the `Zoltan_LB_Partition()` function. A flag returned by the `Zoltan_LB_Partition()` indicates whether a new distribution is required. If so, bubble redistribution is done; otherwise previous distribution is maintained. Unlike initial distribution, the dynamic redistribution involves subfront-subfront communication for transferring bubbles among subfronts.

The pseudocode for bubble distribution described above and the overall load balancing strategy is depicted in Algorithm 1.

4.2. Remapping Interprocess Communication. As discussed in Section 3.1, there are three types of interprocess communication in the fixed mapping simulation. While load balancing strategy does not affect subdomain-subdomain communication, the other two types of communication involving subfront processes are affected as shown in Algorithm 1.

4.2.1. Subdomain-Subfront Communication. Due to the revised bubble assignment scheme for subfront processes, the

```
! Subfront initialization functions
 ...
 ! Initialize Zoltan library
 Zoltan_Initialize()
 Zoltan_handle=Zoltan_Create(Subfront_Comm)
 Zoltan_Set_Param(Zoltan_handle,"LB_METHOD", "RCB")

 ! Initialize bubbles as per fixed-mapping strategy
 Initial_Bubble_Distribution()

 ! Zoltan RCB bubble partitioning, it returns
 ! Number of bubbles to import/export and their IDs,
 ! process IDs for import/export, boolean flag showing
 ! whether partition was updated
 Zoltan_LB_Partition(...)

 ! Initial bubble redistribution and load balancing
 do for each bubble to be exported
     Deallocate bubble memory and data structures
 end do

 do for each bubble to be imported
     Allocate bubble memory and initialize data structures
 end do

 ! Subfront main loop
 do for each timestep
    ! Subfront calculations
    ...
    ! Dynamic load balancing and bubble redistribution
    if iteration==load balancing iteration
         Zoltan_LB_Partition(...)
         if partition updated
             do for each bubble to be exported
                 Send bubble to the appropriate process
                 Deallocate bubble memory and data structures
             end do
             do for each bubble to be imported
                 Allocate bubble memory
                 Receive bubble from appropriate process
                 Initialize bubble data structures
             end do
         end if
    end if
 end do
```

Algorithm 1: Pseudocode for subfront processes emphasizing bubble distribution and load balancing.

subfront-subdomain communication requires remapping. Previously, a subfront process was mapped to predetermined fixed number of subdomain processes as it operates on bubbles belonging to those subdomains only and if the bubbles moved out of those subdomain(s), its ownership is transferred to the corresponding subfront. In the new scheme, the concept of subfront bubble ownership is not based on location in the subdomain. It is instead based on the load balancing output. As a result, bubbles owned by a subfront may be lying in different subdomains. It is even possible that points of a single bubble may be lying in different subdomains with their location constantly changing over the course of the simulation. This means that both subdomain and subfront processes need to dynamically identify subfronts/subdomains which they need to communicate with. In other words, mapping from subfronts to subdomains needs to be *adaptive*. This adaptive subfront-subdomain mapping (adaptive mapping) is achieved as explained below:

(i) Each subfront process iterates over all of its bubble points and, based on the knowledge of subdomain boundaries, it calculates number of points belonging to a particular subdomain and stores this count

Table 1: Example subdomain communication table for subfront i.

Subdomain ID	Subdomain 0	Subdomain 1	Subdomain 2	Subdomain 3
Bubble points	1428	0	13320	0

Table 2: Example overall subdomain-subfront communication table.

Subfront/subdomain ID	Subfront 0	Subfront 1	Subfront 2	Subfront 3
Subdomain 0	1428	0	16768	1550
Subdomain 1	0	13320	0	1428
Subdomain 2	13320	0	0	760
Subdomain 3	0	20	13320	1350

Table 3: Example subfront communication table for subdomain i.

Subfront ID	Subfront 0	Subfront 1	Subfront 2	Subfront 3
Bubble points	1428	0	16768	1550

in a communication table that we will refer to as *subdomain table*. At the same time, it stores points belonging to a particular subdomain in a separate list. An example subdomain table for subfront i with four subdomains is shown in Table 1. It shows that subfront i has 1428 bubble points for subdomain 0, 13320 for subdomain 2, and none for the rest. Subfront i can then use this table to post-MPI send/receive messages for subdomains accordingly.

(ii) Like subfront processes, subdomain processes dynamically need to know which subfront processes they need to communicate with. For this purpose, subdomain tables of all subfronts are combined and updated on all subfront processes using a *Gather All* operation. An example overall communication table for four subfronts and four subdomains is shown in Table 2. As shown, a row of Table 2 basically constitutes the subfront table for subdomain i. To efficiently update the table on corresponding subdomains, each subfront sends rows of overall communication table to respective subdomains using fixed subdomain-subfront mapping discussed in Section 3.1. An example of subfront table for subdomain i is shown in Table 3.

(iii) Using respective subdomain and subfront tables, subfront-subdomain processes post send/receive messages to exchange bubble data.

4.2.2. Subfront-Subfront Communication. In fixed subdomain-subfront mapping multiphase flow simulation, subfront-subfront communication mainly exists for exchanging shared bubble data and bubble ownership transfer among subfronts. In the current strategy, the concept of shared bubbles has been revoked. Instead, all points of a given bubble are owned by the same subfront unless load balancing output demands redistribution of the bubble to a different subfront. This means that subfront-subfront communication for shared bubble data exchange does not exist anymore. Instead, subfront-subfront communication exists for the following purposes:

(i) To update overall subdomain-subfront communication table on all subfronts

(ii) To transfer bubbles among subfront processes for dynamic load balancing.

Unlike fixed mapping, subfront-subfront communication in adaptive mapping approach is more predictable. In fixed mapping approach, number of subfront-subfront messages exchanged per time step is given by $2(s-1)b$, where s is the number of subfronts sharing bubbles and b is the number of shared bubbles [11]. Both s and b are variable quantities depending upon the movement of bubbles during simulation. It is, therefore, hard to predict in advance as to how many messages will be exchanged among subfront processes in a given iteration. Another type of subfront-subfront communication in fixed mapping strategy is for bubble ownership transfer. If n subfronts need to transfer ownership of their bubbles with each sending messages to m fronts, then total number of messages for ownership transfer is given by $\sum_{i=1}^{n} m_i$. Similarly, both n and m are variable quantities and depend upon movement of bubbles during simulation and therefore it is hard to predict when and how many messages will be exchanged for ownership transfer.

In adaptive mapping approach, a number of subfront-subfront messages per iteration for subdomain-subfront communication table update are given by $n_f(n_f-1)$ where n_f is the total number of subfront processes (*Gather All* operation). As n_f is constant during the course of the simulation, a number of subfront-to-subfront messages per iteration are fixed. Like fixed mapping strategy, bubble ownership transfer may be required after a predetermined number of iterations based on the output of load balancing algorithm. Although the expression for the number of messages for ownership transfer remains the same as for fixed mapping strategy, variation in n and m from one load balancing iteration to the next is not expected to be drastic due to incremental nature of geometric partitioning algorithm [27]. Due to fixed number of subfront-subfront messages per iteration and a more predictable number of messages for ownership transfer each load balancing iteration, overall subfront-subfront communication in adaptive mapping strategy is more predictable compared with fixed mapping strategy.

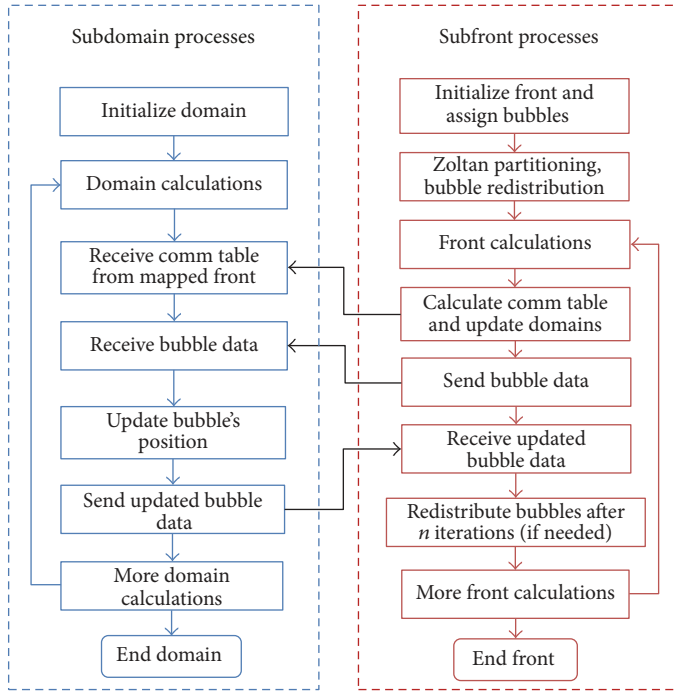


Figure 5: Simplified flowchart of modified parallel multiphase flow simulation. Black arrows indicate interprocess communication.

4.3. Load-Balanced Program Structure. Figure 5 shows the flow chart of modified subdomain and subdomain processes along with remapped communication. As before, the two sets of processes perform their respective process initializations. Bubbles are then assigned to subfront processes as per previous spatial location based approach. Initial redistribution of bubbles based on the Zoltan load balancing output is then performed, followed by computations on the bubbles and update on the subdomain and subfront communication tables. After that, we perform subfront-subdomain bubble data exchange and position update as per new communication scheme. For subdomain processes, intrasubdomain communication and calculations on the Eulerian grid remain unchanged. For subfront processes, bubble distribution is checked after a specified number of iterations and if redistribution is required, bubbles are exchanged among subfront processes accordingly. As in the fixed mapping strategy, all the communications are nonblocking employing communication and computation overlap.

5. Results and Discussion

To carry out performance studies, we use the Cray XC40 (Hazel Hen) supercomputer located at the High-Performance Computing Center, Stuttgart, Germany. Specifications of the supercomputer are listed in Table 4. Details of the software used in the study are listed in Table 5. Bubble radius is chosen such that a bubble spans at least 20 points of the Eulerian grid. All calculations are performed using double precision arithmetic and periodic boundary conditions for all the three directions. For all experiments, two OpenMP processes are used for hyperthreading per MPI process.

Table 4: Specifications of Cray XC40 (Hazel Hen) supercomputer.

CPU	Intel Xeon E5-2680 v3
Number of compute nodes	7712
Sockets per compute node/cores per socket	2/12
Threads per core	2
Clock rate (GHz)	2.5
Memory per node (GB)	128
Shared L3 Cache (MB)	30
Memory bandwidth (GB/s)	68
Interconnect	Cray Aries Dragonfly
Network bandwidth (GB/s)	11.7

Table 5: Specifications of software used.

OS	SUSE Linux Enterprise Server 11
Linux Kernel version	3.0.101-0.47.102-default
Fortran/C compilers	Intel ifort & icc v 17.0.2.174
Cray Trilinos (for Zoltan)	12.10.1.1
PETSc (for Hypre)	3.7.6.0
Cray MPICH	7.6.0

5.1. Strong Scaling. In strong scaling study, we simulate 864 bubbles in $2 \times 2 \times 2$ sized domain with a $512 \times 512 \times 512$ mesh resolution for the Eulerian grid. Bubble distribution pattern is similar to the one shown in Figure 6. The bubble distribution is chosen such that half of the domain is empty while the other half contains bubbles creating 50% load imbalance for subfront processes. Our choice of number of bubbles, grid size, and mesh resolution for Eulerian grid is driven by the

Table 6: Strong scaling inputs and process configurations.

Domain size ($x \times y \times z$)	$2 \times 2 \times 2$
Eulerian grid resolution	$512 \times 512 \times 512$
Number of bubbles	864
Total processors (x, y, z) = subdomain processors ($x \times y \times z$) + subfront processors ($x \times y \times z$)	$2 = 1 \times 1 \times 1 + 1 \times 1 \times 1$
	$4 = 2 \times 1 \times 1 + 2 \times 1 \times 1$
	$8 = 2 \times 2 \times 1 + 2 \times 2 \times 1$
	$16 = 2 \times 2 \times 2 + 2 \times 2 \times 2$
	$32 = 4 \times 2 \times 2 + 4 \times 2 \times 2$
	$64 = 4 \times 4 \times 2 + 4 \times 4 \times 2$
	$128 = 4 \times 4 \times 4 + 4 \times 4 \times 4$
	$256 = 8 \times 4 \times 4 + 8 \times 4 \times 4$
	$512 = 8 \times 8 \times 4 + 8 \times 8 \times 4$
MPI processes/node	16

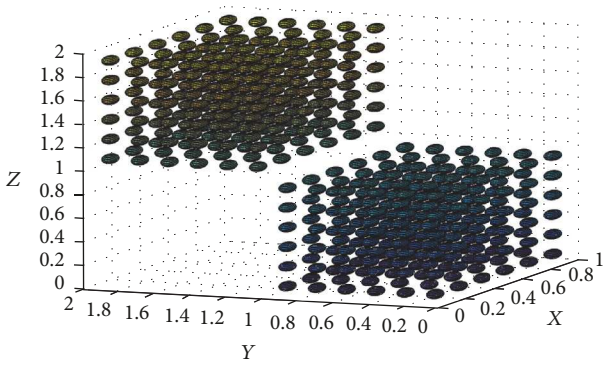


Figure 6: Bubble distribution for $1 \times 2 \times 2$ grid with 432 bubbles. Half of the domain contains bubbles while the rest is empty creating 50% load imbalance for subfront processes.

need to have enough strong scaling configurations without exceeding memory limits of any process or processing node for any of the given configurations. Strong scaling inputs and process configurations are shown in Table 6, where 6, domain size ($x \times y \times z$), refers to volume of the rectangular domain with x, y, and z dimensions in meter. Eulerian grid resolution is number of grid points used to represent the Eulerian grid. Speedup for our load balancing approach (adaptive mapping) is reported against the fixed mapping simulation with the same subdomain and subfront process configurations.

To calculate average iteration time and average speedup for each strong scaling configuration, we performed five experiments per configuration. The results are shown in Figures 7(a) and 7(b), respectively. In Figure 7(b), we notice that, due to variable nature of subfront-subfront communication in fixed mapping, speedup does not show a consistent trend up to 16 + 16 (subdomain + subfront) process configuration. For these process configurations, if the number of shared bubble points is more, we observe a higher speedup for the adaptive mapping version because of the absence of subfront-subfront communication needed for shared bubble exchange in fixed mapping version. On the other hand, if the number of shared bubble points is few or none, the speedup is close to 1. However, from 16 + 16 process configuration onward, we observe higher speedup owing to smaller subdomains and subfronts resulting in an increase in the number of shared bubbles. A speedup of over 5% is achieved with adaptive mapping over fixed mapping for 256 + 256 process configuration.

Although the performance improvement for strong scaling is not very significant, the study shows that the adaptive mapping strategy is always either as good as or better than the fixed mapping strategy for all chosen strong scaling configurations.

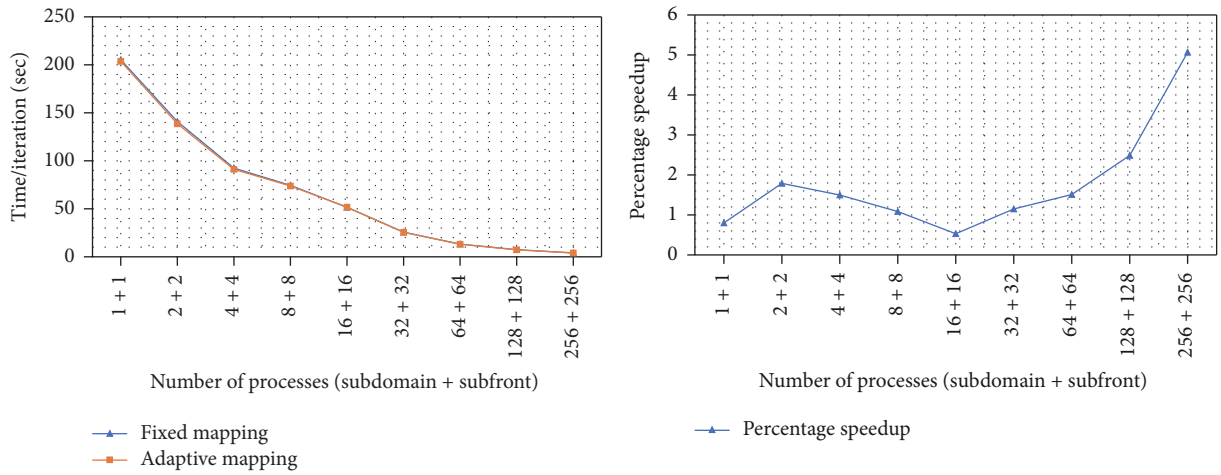
5.2. Weak Scaling. Table 7 shows inputs and process configurations for the weak scaling experiment. To create a load imbalance configuration, a number of bubbles are chosen to be subject to the constraints that fill half of the domain; each bubble spans 20 grid points of the domain along each of the x, y, and z dimensions, and initial distance between any two bubbles is equal to the their radius. This leaves approximately 7 bubbles per subfront processor after load balancing. The Eulerian grid points for each of the subdomain processors are fixed at $64 \times 128 \times 128$ simulating a maximum of 13824 bubbles on 2048 + 2048 processes. To calculate average iteration time and average speedup for each weak scaling configuration, we performed five experiments per configuration. The results are shown in Figures 8(a) and 8(b), respectively. As the domain size and number of bubbles increase, iteration time increases for both fixed and adaptive mapping configurations due to a larger number of messages. However, because of the absence of subfront-subfront communication in adaptive mapping version, time per iteration increases slowly allowing better scaling. As shown in Figure 8(b), a speedup of up to 30% is observed over the fixed mapping at 2048 + 2048 processes simulating 13824 bubbles with a $4 \times 4 \times 8$ domain.

Performance improvement due to the adaptive mapping in weak scaling appears to be higher than the one in strong scaling. However, performance improvement for weak scaling is comparable to strong scaling at the similar process configurations (e.g., 256 + 256). The higher performance improvements are observed on bigger domain sizes and larger process counts. In Section 5.1, our choice for strong scaling configuration is based on the memory capacity requirement in each node. As a result, strong scaling results are likely to improve by the same order as weak scaling for bigger domain sizes, higher number of bubbles, and Eulerian grid points.

5.3. Load Balancing Overhead. Load balancing incurs two types of overheads. The first one includes initial load balancing overhead at the beginning of the simulation while the second one is due to bubble redistribution for dynamic load balancing during the course of simulation. Figure 9 shows comparison of initialization time for fixed and adaptive mapping versions. As evident, there is negligible difference between initialization times for the two versions for strong scaling. This is true because although adaptive mapping version involves bubble distribution/redistribution during initialization, absence of shared bubble data exchange in adaptive mapping compensates for this added overhead.

Table 7: Weak scaling inputs and process configurations.

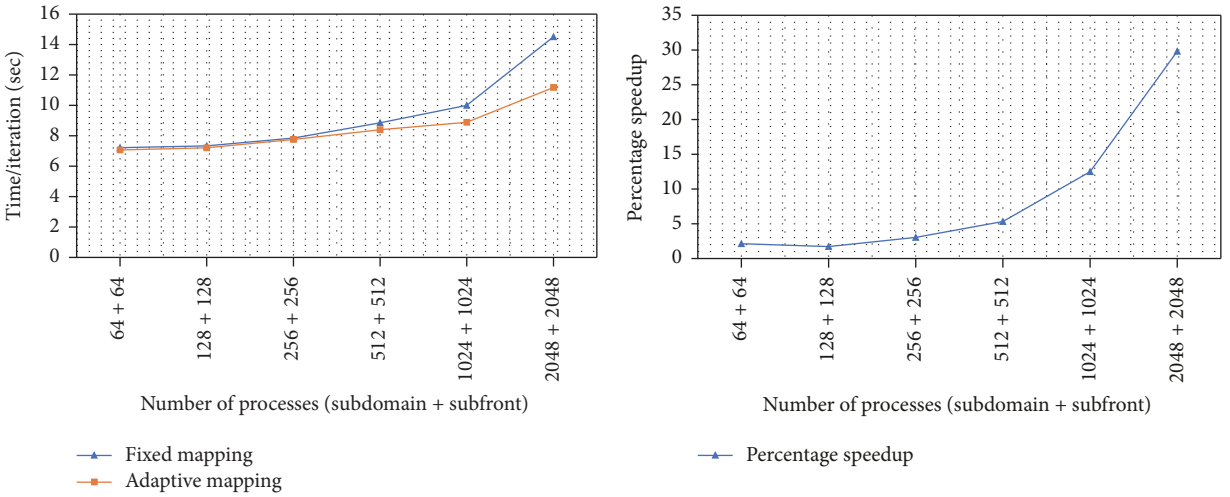
Domain size ($x \times y \times z$)	Mesh resolution	Number of bubbles	Number of processors	Subdomain processors ($x \times y \times z$)	Subfront processors ($x \times y \times z$)
$1 \times 2 \times 2$	$256 \times 512 \times 512$	432	128	$4 \times 4 \times 4 = 64$	$4 \times 4 \times 4 = 64$
$2 \times 2 \times 2$	$512 \times 512 \times 512$	864	256	$8 \times 4 \times 4 = 128$	$8 \times 4 \times 4 = 128$
$2 \times 4 \times 2$	$512 \times 1024 \times 512$	1728	512	$8 \times 8 \times 4 = 256$	$8 \times 8 \times 4 = 256$
$2 \times 4 \times 4$	$512 \times 1024 \times 1024$	3456	1024	$8 \times 8 \times 8 = 512$	$8 \times 8 \times 8 = 512$
$4 \times 4 \times 4$	$1024 \times 1024 \times 1024$	6912	2048	$16 \times 8 \times 8 = 1024$	$16 \times 8 \times 8 = 1024$
$4 \times 4 \times 8$	$1024 \times 1024 \times 2048$	13824	4096	$16 \times 8 \times 16 = 2048$	$16 \times 8 \times 16 = 2048$



(a) Average time per iteration

(b) Average adaptive over fixed mapping speedup

Figure 7: Adaptive versus fixed mapping comparison for strong scaling.



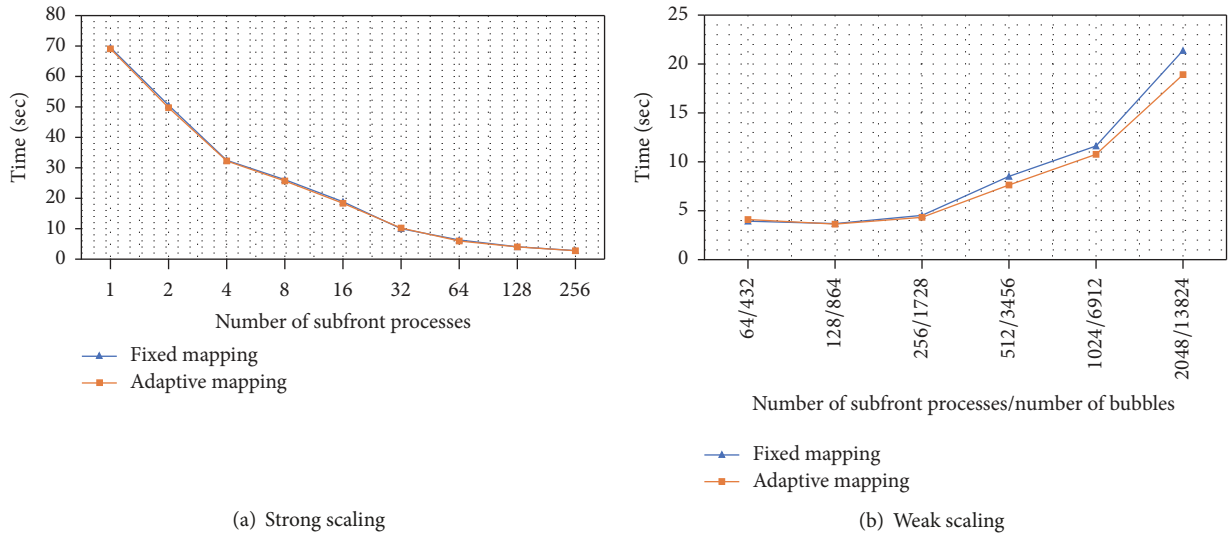
(a) Average time per iteration

(b) Average adaptive over fixed mapping speedup

Figure 8: Adaptive versus fixed mapping comparison for weak scaling.

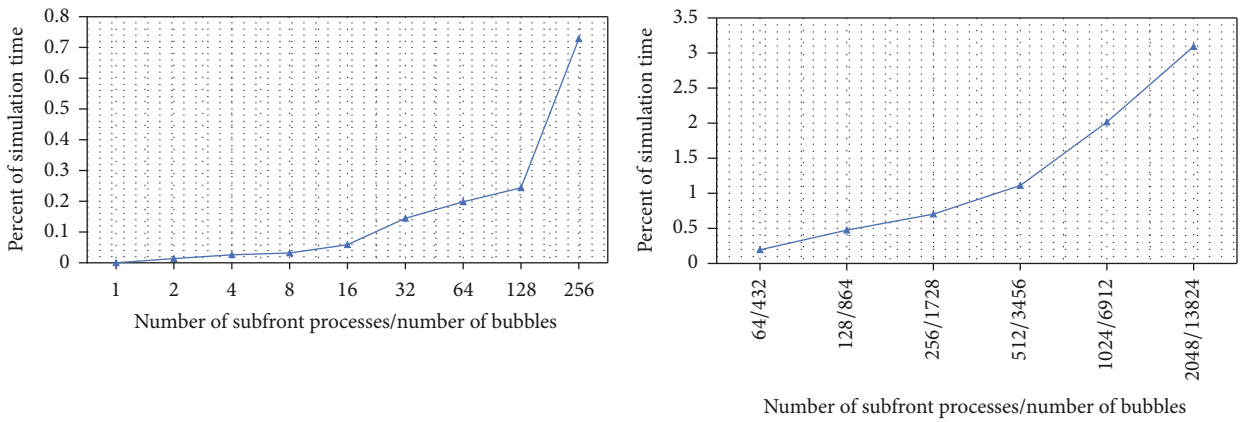
For weak scaling, we notice that as the number of bubbles increases, the shared bubble data exchange overhead in fixed mapping overtakes bubble distribution/redistribution overhead in adaptive mapping resulting in reduced initialization time for adaptive mapping as the problem size increases.

The second type of overhead includes both dynamically detecting the load imbalance and redistributing the load during the course of simulation. Figures 10(a) and 10(b) show the overhead for strong and weak scaling, respectively. Dynamic load balancing is performed after a user specified



(a) Strong scaling (b) Weak scaling

Figure 9: Average simulation initialization time.



(a) Strong scaling (b) Weak scaling

Figure 10: Average dynamic load balancing overhead (as percentage of simulation time).

number of iterations which for the purpose of this study has been set to five. We plot the overhead as percentage of total simulation time. Figure 10(a) shows that dynamic load balancing overhead for strong scaling is less than 0.8% for all process configurations. For weak scaling, maximum dynamic load balancing overhead is a little over 3% for the largest problem size and is 2% or less for all other problem sizes simulated in this study. Overall, the load balancing with adaptive mapping does not introduce too much overhead. Even for cases where it introduces an overhead of more than 1%, the overall execution time improves as compared to the fixed mapping.

6. Conclusions

In this work, we developed a load balancing strategy for a front-tracking method based parallel multiphase flow simulation. Instead of fixed load assignment of movable Lagrangian grid (bubbles) to processors, we perform dynamic and adaptive load balancing resulting in uniform load distribution. We also remapped the interprocess communication resulting in a more predictable variable mapping compared with the existing fixed mapping strategy. Our load balancing strategy is dynamic in nature as we dynamically check for load imbalance during the course of simulation and perform load balancing accordingly.

To check the effectiveness of our strategy, we performed both strong and weak scaling studies on Cray XC40 (Hazel Hen) supercomputer. The strong scaling experiments showed a moderate speedup of over 5% for adaptive mapping over the fixed mapping baseline with same process configuration simulating 864 bubbles; however, considerable performance improvement was observed as the problem size was increased. For instance, in weak scaling experiment of simulating 13824 bubbles with a total of 4096 processes in a domain of grid size $1024 \times 1024 \times 2048$, a speedup of up to 30% was observed for adaptive mapping over fixed mapping with same process configuration. Finally, we

performed analysis of load balancing overheads and observed that, even in their presence, overall execution time improves as compared to the fixed mapping.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors are supported by the Scientific and Technological Research Council of Turkey (TUBITAK), Grant no. 215E193. They acknowledge PRACE for awarding them access to the Hazel Hen supercomputer in Germany.

References

[1] C. Brennen, *Fundamentals of Multiphase Flow*, Cambridge University Press, Cambridge, UK, 2005.

[2] G. Tryggvason, R. Scardovelli, and S. Zaleski, *Direct Numerical Simulations of Gas-Liquid Multiphase Flows*, Cambridge University Press, Cambridge, UK, 2011.

[3] J. Pilliod and E. G. Puckett, "Second-order accurate volume-of-fluid algorithms for tracking material interfaces," *Journal of Computational Physics*, vol. 199, no. 2, pp. 465–502, 2004.

[4] W. J. Rider and D. B. Kothe, "Reconstructing volume tracking," *Journal of Computational Physics*, vol. 141, no. 2, pp. 112–152, 1998.

[5] V. E. Badalassi, H. D. Ceniceros, and S. Banerjee, "Computation of multiphase systems with phase field models," *Journal of Computational Physics*, vol. 190, no. 2, pp. 371–397, 2003.

[6] D. M. Anderson, G. B. McFadden, and A. A. Wheeler, "Diffuse-interface methods in fluid mechanics," *Annual Review of Fluid Mechanics*, vol. 30, no. 1, pp. 139–165, 1998.

[7] M. Sussman, P. Smereka, and S. Osher, "A level set approach for computing solutions to incompressible two-phase flow," *Journal of Computational Physics*, vol. 114, no. 1, pp. 146–159, 1994.

[8] E. Olsson and G. Kreiss, "A conservative level set method for two phase flow," *Journal of Computational Physics*, vol. 210, no. 1, pp. 225–246, 2005.

[9] G. Tryggvason, B. Bunner, A. Esmaeeli et al., "A front-tracking method for the computations of multiphase flow," *Journal of Computational Physics*, vol. 169, no. 2, pp. 708–759, 2001.

[10] J. Glimm and O. A. McBryan, "A computational model for interfaces," *Advances in Applied Mathematics*, vol. 6, no. 4, pp. 422–435, 1985.

[11] M. N. Farooqi, D. Izbassarov, M. Muradoğlu, and D. Unat, "Communication analysis and optimization of 3D front tracking method for multiphase flow simulations," *International Journal of High Performance Computing Applications*, 2017.

[12] M. J. Berger and S. H. Bokhari, "A partitioning strategy for nonuniform problems on multiprocessors," *IEEE Transactions on Computers*, vol. C-36, no. 5, pp. 570–580, 1987.

[13] B. Hendrickson and K. Devine, "Dynamic load balancing in computational mechanics," *Computer Methods Applied Mechanics and Engineering*, vol. 184, no. 2-4, pp. 485–500, 2000.

[14] J. Watts, M. Rieffel, and S. Taylor, "A load balancing technique for multiphase computations," in *Proceedings of High Performance Computing' 97*, pp. 15–20, 1997.

[15] M. H. Willebeek-LeMair and A. P. Reeves, "Strategies for dynamic load balancing on highly parallel computers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 9, pp. 979–993, 1993.

[16] M. Rieffel, S. Taylor, J. Watts, and S. Shankar, "Concurrent simulation of plasma reactors," in *Proceedings of the in Proceedings of High Performance Computing'97*, pp. 163–168, 1997.

[17] F. Ham, S. Apte, G. Iaccarino, X. Wu, and M. Herrmann, "Unstructured LES of reacting multiphase ows in realistic gas turbine combustors," Tech. Rep., Minnesota University Minneapolis, Minnesota, Minn, USA, 2003.

[18] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 359–392, 1999.

[19] R. Borrell, L. Jofre, O. Lehmkuhl, and J. Castro, "Parallelization strategy for the volume-of-fluid method on unstructured meshes," *Procedia Engineering*, vol. 61, pp. 198–203, 2013.

[20] G. Agbaglah, S. Delaux, D. Fuster et al., "Parallel simulation of multiphase ows using octree adaptivity and the volume-of-fluid method," *Comptes Rendus Mecanique*, vol. 339, no. 2-3, pp. 194–207, 2011.

[21] A. Ikebata and F. Xiao, "GPU-accelerated large-scale simulations of interfacial multiphase fluids for real-case applications," *Computers and Fluids*, vol. 141, pp. 235–249, 2016.

[22] M. Aldinucci, M. Danelutto, M. Drocco et al., "A parallel pattern for iterative stencil + reduce," *The Journal of Supercomputing*, 2016.

[23] M. Aldinucci, G. P. Pezzi, M. Drocco, C. Spampinato, and M. Torquati, "Parallel visual data restoration on multi-GPGPUs using stencil-reduce pattern," *International Journal of High Performance Computing Applications*, vol. 29, no. 4, pp. 461–472, 2015.

[24] G. Houzeaux, M. Garcia-Gasulla, J. C. Cajas, A. Artigues, E. Olivares, and J. Labarta, "Dynamic load balance applied to particle transport in fluids," *International Journal of Computational Fluid Dynamics*, vol. 30, no. 6, pp. 408–418, 2016.

[25] M. Garcia, J. Labarta, and J. Corbalan, "Hints to improve automatic load balancing with LeWI for hybrid applications," *Journal of Parallel and Distributed Computing*, vol. 74, no. 9, pp. 2781–2794, 2014.

[26] S. O. Unverdi and G. Tryggvason, "A front-tracking method for viscous, incompressible, multifluid flows," *Journal of Computational Physics*, vol. 100, no. 1, pp. 25–37, 1992.

[27] K. D. Devine, E. G. Boman, and G. Karypis, "Partitioning and load balancing for emerging parallel applications and architectures," in *Parallel Processing for Scientific Computing*, M. A. Heroux, P. Raghavan, and H. D. Simon, Eds., chapter 6, pp. 99–126, Society for Industrial and Applied Mathematics, Pennsylvania, Pa, USA, 2006.

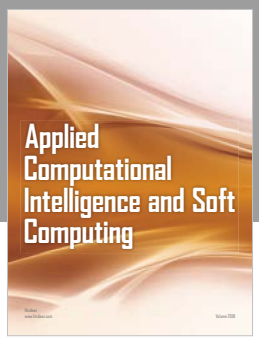
[28] B. Maerten, D. Roose, A. Basermann, J. Fingberg, and G. Lonsdale, "DRAMA: A library for parallel dynamic load balancing of finite element applications," in *Proceedings of the European Conference on Parallel Processing*, pp. 313–331, 1999.

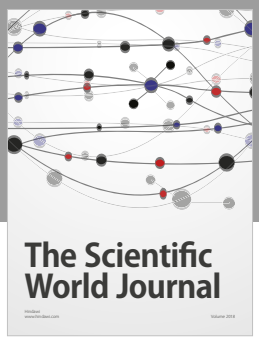
[29] A. Basermann, J. Clinckemaillie, T. Coupez et al., "Dynamic load-balancing of finite element applications with the DRAMA library," *Applied Mathematical Modelling*, vol. 25, no. 2, pp. 83–98, 2000.

[30] L. Oliker and R. Biswas, "PLUM: parallel load balancing for adaptive unstructured meshes," *Journal of Parallel and Distributed Computing*, vol. 52, no. 2, pp. 150–177, 1998.

[31] E. G. Boman, Ü. V. Çatalyürek, C. Chevalier, and K. D. Devine, "The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: partitioning, ordering and coloring," *Scientific Programming*, vol. 20, no. 2, pp. 129–150, 2012.

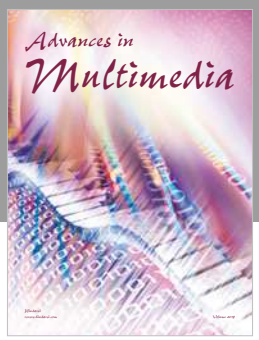
[32] K. D. Devine, E. G. Boman, R. T. Heaphy, R. H. Bisseling, and U. V. Catalyurek, "Parallel hypergraph partitioning for scientific computing," in *Proceedings of the 20th IEEE International Parallel Distributed Processing Symposium*, p. 10, 2006.

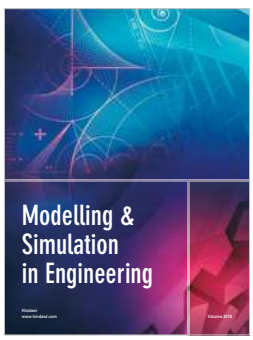


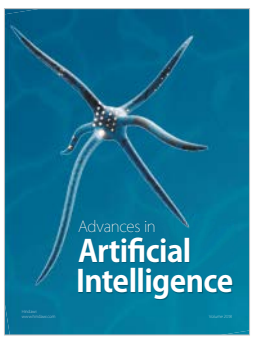




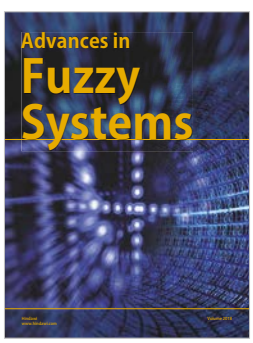












Hindawi
Submit your manuscripts at
www.hindawi.com



