ABSTRACT
Cavitating bubbly flows are encountered in many engineering problems involving propellers, pumps, valves, ultrasonic biomedical applications, etc. In this contribution an OpenMP parallelized Euler-Lagrange model of two-phase flow problems and cavitation is presented. The two-phase medium is treated as a continuum and solved on an Eulerian grid, while the discrete bubbles are tracked in a Lagrangian fashion with their dynamics computed. The intimate coupling between the two description levels is realized through the local void fraction, which is computed from the instantaneous bubble volumes and locations, and provides the continuum properties. Since, in practice, any such flows will involve large numbers of bubbles, schemes for significant speedup are needed to reduce computation times. We present here a shared-memory parallelization scheme combining domain decomposition for the continuum domain and number decomposition for the bubbles; both selected to realize maximum speed up and good load balance. The Eulerian computational domain is subdivided based on geometry into several subdomains, while for the Lagrangian computations, the bubbles are subdivided based on their indices into several subsets. The number of fluid subdomains and bubble subsets are matched with the number of CPU cores available in a share-memory system. Computation of the continuum solution and the bubble dynamics proceeds sequentially. During each computation time step, all selected OpenMP threads are first used to evolve the fluid solution, with each handling one subdomain. Upon completion, the OpenMP threads selected for the Lagrangian solution are then used to execute the bubble computations. All data exchanges are executed through the shared memory. Extra steps are taken to localize the memory access pattern to minimize non-local data fetch latency, since severe performance penalty may occur on a Non-Uniform Memory Architecture multiprocessing system where thread access to non-local memory is much slower than to local memory.

This parallelization scheme is illustrated on a typical non-uniform bubbly flow problem, cloud bubble dynamics near a rigid wall driven by an imposed pressure function.

INTRODUCTION
Cavitating bubbly flows occur in various engineering problems such as in ultrasonic biomedical applications, cavitating jets for biomass treatments, unsteady sheet cavities on propeller blades, etc. [1-6]. Numerical modeling of such problems is very challenging, since it involves bubble-bubble, bubble-flow, and bubble-wall interactions [5]. In addition the problem involves multi-scale physics ranging from micron-scale individual bubbles to meter-scale flow field (e.g., propeller blade size). In between, there are very rich and complex meso-scale phenomena such as at the scale of bubble clouds, where the bubbles act collectively [7-12].

Fully resolved methods such as Direct Numerical Simulation (DNS) or the Boundary Element Method (BEM)
can provide, within their limiting assumptions, details at several scales of interests with impressive progress reported [e.g., 11, 13-15], but are limited to detail problems and not applications due to high computational cost. In practical engineering applications, cavitating bubbly flows are usually modeled using one of several approaches: equivalent homogeneous continuum models [3,7,16], Eulerian two-fluid models [17-20], or Eulerian-Lagrangian approaches wherein the bubbles are treated as discrete elements. The former two are based on volume or ensemble averaged approximations thus suitable for cases where the bubbles are much smaller than the characteristic lengths associated with the overall dynamics of the bubbly mixture. They become inadequate when the scales associated with the individual bubbles are comparable to the mixture flow scales. Eulerian-Lagrangian methods have been developed to treat these cases (e.g., [21-27]), where the carrier fluid is modeled using continuum fluid methods (Eulerian approach) and bubbles are tracked individually (Lagrangian approach). These methods have shown intrinsic advantages owing to their capability of including the physics across scales down to each individual bubble [12]. However, the number of bubbles in actual industrial systems is so large that the computation cost prevents full scale applications. Therefore, efficient parallelization schemes for significant speedup are needed to reduce computation times.

In this paper, we present an OpenMP shared-memory parallelization for Eulerian-Lagrangian modeling of cavitating bubbly flows taking advantage of multi-core multiprocessing computer systems. These systems usually consist of several multi-core CPUs with a large shared memory pool. In principal, an efficient parallelization should distribute the computation load of both Eulerian and Lagrangian phases as evenly as possible through all the CPU cores (i.e., OpenMP threads) to reach an optimum load balance. However, bubbles do not remain uniformly distributed in the fluid domain and are not suitable for domain decomposition. Therefore, this study employs a hybrid parallelization scheme combining domain decomposition for the Eulerian fluid domain and number decomposition for the Lagrangian bubbles. A similar hybrid decomposition method has also been reported in OpenMP parallelization of Discrete Element Modeling of particulate flows with ideal load balancing [28]. However, a drawback of mixing different decomposition methods for is the high overhead involved in non-local data fetching when exchanging information between fluid and particle phases during the two-way coupling. Overhead may become considerable and may cancel out parallelization speed up on a Non-Uniform Memory Architecture (NUMA) multiprocessing system. Therefore, another major effort in this paper is to address the locality of memory access and reduce non-local data fetch latency, thus minimize any performance penalties.

The paper is organized as follows. The main features of the fully coupled Euler-Lagrangian model are summarized in the next section. They are followed by a description of the OpenMP parallelization scheme. Scaling and efficiency analysis of the scheme are then tested on the problem of the dynamics of microbubble clouds near a rigid wall [12]. This case is selected as it is numerically challenging and presents a typical non-uniform bubbly flow configuration where bubbles are confined to a small spherical region in the much larger fluid domain. A comparison of the speed up between non-local and localized data access patterns is then provided. Finally, a summary of the findings is presented.

**Eulerian-Lagrangian Model**

The 3D Eulerian-Lagrangian model employed in this study has been extensively validated and documented in our previous studies. These included the investigation of the effects of a propeller flow on bubble size distribution in water [29,30], the modeling of propeller tip vortex cavitation inception [30,31], simulation of the bubbly flow in a bubble augmented propulsor [32], bubble entrainment in plunging jets [33] and wave propagation in bubbly media [25-27], etc. The present paper addresses acceleration of the computations through parallelization and does not consider additional model development. Therefore, below we only provide a brief summary of the key features of the model and concentrate on the parallelization aspects. Readers interested in more details on modeling aspects may refer to the references provided above.

**Viscous Mixture Flow Solver**

The unsteady Navier-Stokes equations for a liquid-gas mixture are as follows:

$$\frac{\partial \rho_m}{\partial t} + \frac{\partial \rho_m u_j}{\partial x_j} = 0,$$

(1)

$$\frac{\partial \rho_m u_j}{\partial t} + \frac{\partial \rho_m u_i u_j}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j} \left[ \rho_m \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \right],$$

(2)

where $\rho_m$ is the mixture density, $u$ its velocity, $p$ its pressure, $\nu_m$ its kinematic viscosity. The mixture density and viscosity can be related to the liquid (subscript ‘l’) and gas (subscript ‘g’) properties by:

$$\rho_m = (1-\alpha) \rho_l + \alpha \rho_g,$$

$$\nu_m = (1-\alpha) \nu_l + \alpha \nu_g,$$

(3)

where $\alpha$ is the gas volume fraction.

Equations (1) and (2) are solved based on the artificial-compressibility method [34], along with a Newton iterative pseudo-time stepping procedure (within each physical time step) to obtain a time-accurate solution. The numerical scheme uses a finite volume formulation, with a first-order Euler implicit difference formula applied to the time derivatives and a flux-difference splitting scheme based on Roe’s method [35] and a van Leer’s MUSCL method [36] for spatial differencing of convection terms. A second-order central differencing is used for the viscous terms.

**Lagrangian Modeling of Dispersed Phase**

For the bubbles dispersed in the medium, we use a Lagrangian bubble tracking method. All bubbles are treated as point sources to account for volume change and dipoles to account for translation. The bubble volume variations are
obtained using a modified Rayleigh-Plesset-Keller-Herring equation [1] which accounts for the surrounding medium compressibility and non-uniform pressure field through the Surface Averaged Pressure (SAP) model [5, 23]:

\[
\left(1 - \frac{R}{c_m}\right) - \frac{2}{3} \left(1 - \frac{R}{3c_m}\right) - \frac{1}{2} \left(1 - \frac{R}{c_m}\right) \frac{\dot{R}}{R} = \frac{(u_{en} - u_{b})^2}{4} - \frac{\dot{R}}{R} + \frac{1}{\rho_m} \left[ 1 + \frac{\dot{R}}{c_m} - \frac{R}{c_m} \frac{dR}{dt} \right] \left[ p_{\rho} + p_{g} - \frac{2\gamma}{R} - 4\mu_m \frac{\dot{R}}{R} \right].
\]

(4)

In the above equation \( R \) is the bubble radius and the dots represent time differentiation. \( \mu_m \) is the two-phase medium dynamic viscosity, \( p_\rho \) is the liquid vapor pressure, \( p_g \) is the initial gas pressure in the bubble. \( k \) is the gas polytropic constant and equals 1.4 under adiabatic conditions and 1 under isothermal assumption, and \( \gamma \) is the water dynamic viscosity, and \( c_m \) is the local sound speed in the mixture, which is related to the local void fraction, \( \alpha \), by

\[
c_{\alpha} = \frac{p}{\rho L} \left[ 1 + \frac{\alpha}{1 - \alpha} \right].
\]

(5)

where \( c_{\alpha} \) is the sound speed in pure liquid [3]. \( p_{en} \) and \( u_{en} \) are the mixture pressures and velocities averaged over the bubble surface and \( u_b \) is the bubble center translation velocity.

The introduction in Equation (4) of \((u_{en} - u_{b})^2/4\) is to account for slip between the bubble and the host medium. The average terms, \( p_{en} \) and \( u_{en} \), are to account for non-uniform pressures and velocities along the bubble surface. The use of \( p_{en} \) results in a major improvement over classical models, which use the pressure at the bubble center.

This approach has been validated in our previous studies [5, 23, 30]. For instance, in the simulation of the dynamics of a bubble captured by a vortex, the conventional spherical model over-predicts significantly the bubble size versus time when using the pressure at the bubble center. On the other hand, using Equation (4) enables a more realistic evaluation of the bubble dynamics with the solution agreeing very well with a fully resolved moving grid method [30].

The bubble trajectory is obtained using the motion equation of Johnson, & Hsieh [37]:

\[
\frac{du_b}{dt} = -\frac{3}{\rho_m} \nabla p - 2g + \frac{3C_D}{4R} (u_{en} - u_b) \left| u_{en} - u_b \right| + \frac{3C_L}{2\pi R} \sqrt{\frac{\mu_m}{\rho}} \left( (u_{en} - u_b) \times \Omega \right) + \frac{3}{R} (u_{en} - u_b) \dot{R},
\]

(6)

where \( \rho_b \) is the density of the gas in the bubble, \( A_b \) is the bubble projected area, and \( V_b \) its volume. \( \Omega \) is the local vorticity, \( C_L \) is the lift coefficient [38]. \( C_D \) is the drag coefficient defined as a function of the bubble Reynolds number, \( R_{eb} \) following [39]:

\[
C_D = \frac{24}{R_{eb}} \left( 1 + 0.197R_{eb}^{0.63} + 2.6 \times 10^{-3}R_{eb}^{1.38} \right)
\]

(7)

**Eulerian-Lagrangian Coupling**

The two-way coupling between the Eulerian continuum-based model and the Lagrangian discrete bubble model is realized as follows [26, 27]:

- The bubble dynamics and motion of the individual bubbles in the flow field are controlled by the two-phase medium local properties and gradients,
- The local properties of the mixture are determined by the bubble size and distribution, and the void fractions and local densities are determined by the instantaneous bubble volume and distributions,
- The mixture flow field has an evolving mixture density in space and time and satisfies mass and momentum conservation.

The key of this coupling scheme is the deduction of the void fraction from the instantaneous bubble sizes and locations. In our previous studies, the void fraction was obtained by dividing in each computational cell the total volume of the bubbles in the cell by the cell volume. We have found that scheme not totally satisfactory because of the requirement of many bubbles in each cell and also mainly because of the non-continuous character of \( \alpha \) across cells resulting in difficulties with differentiation [25]. To smooth the discontinuities and improve stability, we have implemented a Gaussian distribution scheme to smoothly “spread” each bubble “void” effect across neighboring cells. This scheme has been found to significantly increase numerical stability and to enable handling of high void bubbly flow simulations [12, 26-27]. The void fraction is obtained using the following expression:

\[
\alpha_i = \sum_{j=1}^{N_i} \sum_{k=1}^{N_{cell}} f_{i,j} V_{i,k}^{b},
\]

(8)

where \( V_{i,k}^{b} \) and \( V_{i,k}^{cell} \) are the volumes of bubble, \( j \), and a cell, \( k \), respectively. \( N_i \) is the number of bubbles which are in the “influence range” to a cell \( i \), \( N_{cell} \) is the number of cells “influenced” by a bubble \( j \). And \( f_{i,j} \) is the weight factor of bubble \( j \) contributing to a cell \( i \) which gradually decays to zero as the distance between the bubble and cell increases. Here the Gaussian distribution function is used as illustrated in Figure 1 [12, 26-27].
PARALLELIZATION ALGORITHM & IMPLEMENTATION

Algorithm and Program Procedure

Some multi-core, multiprocessing systems have a shared memory pool consisting of multiple, multi-core CPUs. For example, DYNA-OMP, one of DYNAFLOW’s multiprocessing workstations, has 4 Twelve-core AMD Opteron 2.80 GHz CPUs with 128 GB shared memory. A shared-memory OpenMP parallelization scheme was implemented on DYNA-OMP to parallelize our Eulerian/Lagrangian coupled model.

As illustrated in Figure 2, a scheme combining domain decomposition for the continuum domain and number decomposition for the bubbles is used. Both decompositions are selected to realize maximum speed up and good load balance.

- The Eulerian fluid domain is geometrically subdivided into subdomains with the continuum computations in each subdomain handled by an OpenMP thread running on one CPU processor core.
- The bubbles are also subdivided, based on their indices, into several subsets with each handled by an OpenMP thread using a single CPU processor core.

The overall flow chart for implementation of the hybrid parallelization algorithm is shown in Figure 3. The algorithm shows that the computations of the continuum solution and the bubble dynamics proceed sequentially. During each computation time step, all selected OpenMP threads (maximum thread number limited by the number of CPU cores available) are first used to execute the continuum computations, with each handling one subdomain. Upon completion, the OpenMP threads selected for the Lagrangian computations are then used to obtain the bubble dynamics. This procedure is repeated for each physical time step until the final step is reached.

OpenMP Implementation

The advantage of domain decomposition for Eulerian fluid solver is that all cells in a given subdomain (assigned to one thread) are stored in contiguous memory addresses. This is very favorable for memory access since continuum cells need to interact only with their neighboring cells. Only the small portion of continuum cells near subdomain borders require information from cells located in different subdomains and have to fetch the data from the global shared memory. The OpenMP parallelization is realized based on a multi-block structure of the code. This is done by enclosing the block (i.e., subdomain) loops with OpenMP directives: “!omp parallel do ... !omp end parallel”.

Similarly, the Lagrangian bubble computations are done by enclosing the loops of bubble dynamics and motion computations with similar directives. The bubbles are divided into a multitude of subsets based on their indexes with each handled independently by a thread processor. Each thread can efficiently access a contiguous global memory space. Therefore, the OpenMP implementation for either the Eulerian or the Lagrangian component alone is straightforward.
However, an efficient parallel implementation for the coupling between the phases, in particular the void fraction computations (Eq. (8)) requires careful consideration for data access pattern. Difficulty stems from the two different decomposition methods applied to the continuum and to the bubbles. Non-local data fetching is necessary since bubbles located in a continuum computation cell may belong several different processor cores. This necessitates irregular, remote data fetch, which result in performance penalty. On a Non-Uniform Memory Architecture (NUMA) multiprocessing system especially, thread access to non-local memory is much slower than to local memory. This high-latency data access mode creates very high overhead, which could cancel out any gains from load balancing.

![Flow chart for the OpenMP parallelized Eulerian-Lagrangian coupled two-phase bubbly flow model.](START)

![Initialize Continuum Computations](Initialization of Continuum Computations
Read inputs, initialize fluid variables/arrays)

![Initialize Bubble Computations](Initialization of Bubble Computations
Read inputs, initialize bubble variables/arrays)

![Enter OpenMP Region](Enter OpenMP Region
Decompose subdomains & assigns them to threads)

![Update Mixture Properties](Update Mixture Properties
Apply Boundary Conditions
Exchange Information between Subdomains)

![Solve Mass and Momentum Equations](Solve Mass and Momentum Equations
Converged ?
No
Yes
Enter OpenMP Region
Decompose subsets & assigns them to threads)

![Interpolate Fluid Quantities on Bubbles](Interpolate Fluid Quantities on Bubbles
Solve Bubble Dynamics & Motion Equations
Converged ?
No
Yes
Exit OpenMP Region)

![Compute Void Fraction](Compute Void Fraction
Solve Bubble Dynamics & Motion Equations
Exit OpenMP Region)

![Output](Output
Finished ?
No
Yes
END)

![Figure 3](START)

![Figure 4](START
Initialization of Continuum Computations
Read inputs, initialize fluid variables/arrays
Converged ?
No
Yes
Initialization of Bubble Computations
Read inputs, initialize bubble variables/arrays
Apply Boundary Conditions
Exchange Information between Subdomains
Enter OpenMP Region
Decompose subdomains & assigns them to threads
Solve Mass and Momentum Equations
Converged ?
No
Yes
Exit OpenMP Region
Enter OpenMP Region
Decompose subsets & assigns them to threads
Interpolate Fluid Quantities on Bubbles
Solve Bubble Dynamics & Motion Equations
Converged ?
No
Yes
Exit OpenMP Region
Compute Void Fraction
Exit OpenMP Region
Output
Finished ?
No
Yes
END)

**Figure 4.** A pseudo code of the void fraction computation illustrating the conversion from remote data access (Left) into localized data access (Right). Here, *nbub* is the total bubble number, *void* and *void_tmp* are the global and local void fraction arrays. *(f_min, f_max), (j_min, j_max)* and *(k_min, k_max)* correspond to the bubble’s void effect along the x, y, z direction, respectively and *(SD_min, SD_max)* means the subdomain range.

To overcome this, extra steps are taken to localize the memory access pattern to minimize data fetch latency. This is illustrated with the pseudo code of void fraction computation in Figure 4. The code stores the void fraction in a global array, “void”. Following Equation (8), for a given cell indexed by *(i,j,k,m)* (where, *i,j,k* are the indices along x, y, z direction, and *m* is the block index), multiple bubbles located inside and around the cell contribute void effects to it. To avoid remote data access (e.g. have each bubbles access the global memory address corresponding to *void*(i,j,k,m) and modify it by adding the void contribution) a temporal array, “void_tmp” is created under each thread and temporarily stores the void contribution of all the bubbles assigned to that thread (Figure 4b). Once the
bubble loop is completed by all threads, the void values stored in the “void_tmp” arrays are summed and transferred to the global array “void”. The temporal arrays are then emptied to release memory. At the beginning of each loop these temporal array must be allocated independently under each thread in order to ensure the locality of data placement by following the “first touch policy” [28]. According to this policy, array data is stored in a memory local to the thread which allocates and initializes the array.

**TEST SIMULATION & RESULTS**

To evaluate the developed OpenMP parallelized Euler-Lagrange model, we apply it to simulate the dynamics of a cloud of microbubbles located near a rigid wall and subjected to a sinusoidal pressure time variation imposed on the computational domain far field boundaries [12]. This is a numerically challenging problem since it involves violent bubble volume changes and strong coupling effects fed back to the flow [3,7,11,12]. Also it is a typical non-uniform void fraction distribution since the bubbles are only present in a small space region while the resolved fluid domain extends in the 3D space much far away than the initial cloud volume. Figure 5 illustrates the dynamics for a bubble cloud of initial radius 1.5mm, consisting of initially mono-dispersed bubbles of radius 50 µm and an average void fraction of 5.0%. This cloud was excited by an imposed sinusoidal pressure variations at infinity with an amplitude of 1.5 atm and a frequency of 10 kHz. The frequency was selected to match the cloud natural frequency [3]. As known from previous studies [12] under this frequency-tuned condition, a collective strongly coupled cluster behavior occurs as the cloud experiences a cascading collapse with the bubbles farthest away from the wall collapsing first and the nearest ones collapsing last cumulating the energy. This results in an extremely high pressure peak at the wall at the end of cloud collapse, as illustrated in the right panel of Figure 5. More simulation details and result analysis can be found in [12] and are not repeated here for brevity. We focus instead on the major objective of this paper, i.e., parallelization efficiency tests and analysis.

![Figure 5: A snapshot of pressures and velocity vectors for a bubble cloud near a wall at the end of its collapse following sinusoidal pressure excitation with a frequency matching the cloud natural frequency (Left). Pressure histories monitored at three locations on the wall during the cloud dynamics (Right) [12].](image)

**Work Load Decomposition and Balancing**

Figure 6 shows how the workload for a case with 65k Eulerian grid and 170k Lagrangian bubbles is distributed, with different colors corresponding to different OpenMP threads. Since the mesh is finer in the cloud region than away from it, the corresponding subdomains are smaller than elsewhere to ensure that each subdomain contains the same number of continuum cells. On the other hand, bubbles are evenly subdivided into subsets of equal number, independent of their locations. Therefore, a good load balance is achieved through this decomposition.

**Scaling Tests and Speedup Analysis**

To examine the scalability of the developed scheme, we tested the code for the same bubble cloud shown above on an Intel Xeon CPU node of a Linux cluster. This node has dual six-core 3.00 GHz Xeon CPUs and a total of 12Gigabytes shared memory. Figure 7 (top) shows the wall time per time step of computation and the speed-up factor (bottom) when varying the number of OpenMP threads. Here a time step includes both the Eulerian and Lagrangian solution and the coupling through void fraction computation. It is seen that The wall time per time step is seen to drop from 923s to about 81.6s when the thread number increases from 1 to 12. The speed-up factor is linear and nearly ideal up to 12 threads as shown in Figure 7 (bottom).
Comparison between Non-local and Localized Memory Accesses

To quantify the gains from the procedure of data access optimization for void fraction computation, the speed up factors for each call of void fraction subroutine are compared in Figure 8 between non-local data access and optimized local data access. The speedup is seen to gradually increase as more threads are used. With 12 threads a 17% improvement can be seen.

For the purpose of further demonstrating the portability of this optimization procedure, similar tests were done on a more challenging Non-Uniform Memory Architecture (NUMA) multiprocessing system, DYNA-OMP. Each of these 4 CPUs in DYNA-OMP consists of 2 NUMA units with each having its own local memory. Thread access to memory belonging to a different NUMA unit has a high-latency and is very inefficient. The corresponding overhead may result in canceling out the acceleration achieved by OpenMP parallelization. This can be clearly observed in the top plot of Figure 9, which shows that if the localization of data access is not used increasing the number of threads over 6 may increase the cost of the void fraction computation. The number 6 is critical because beyond 6 threads more than two NUMA units are involved and remote fetch data becomes necessary. On the contrary, after applying the localized data access optimization significant speed up can be achieved with increased thread numbers. Figure 9 shows that by increasing the number of threads from 1 to 48, the speed up factor grows to 44, a parallelization efficiency of about 92%. The above examples also indicate good portability of the developed parallelization scheme since good speedups were achieved with both Intel and AMD multi-core multiprocessing systems, which use very different architectures including core number, memory size and memory types.

CONCLUSION

An OpenMP shared-memory parallelized 3D Euler-Lagrange model was developed for two-phase and cavitating bubbly flow problems. This parallelization scheme combines domain decomposition for the continuum domain and number decomposition for the bubbles to guarantee a good load balance, even for non-uniform bubble distributions. The executions of the Eulerian and Lagrangian solver proceed sequentially. All available OpenMP threads are first used to execute the continuum computations with each handling one subdomain. The same are then used to execute the bubble computations with each thread handling one bubble subset. All data exchanges are executed through accessing the shared memory. However, special consideration is given in the programming to localize as much as possible memory access through use of temporary local memory in order to minimize remote data fetch latency.

Scaling tests were conducted using the problem of a bubble cloud dynamics near a rigid wall. A nearly ideal speed up could be achieved on both a 12-core Intel Xeon Node and a NUMA 48-core AMD Opteron workstation.
REFERENCES


