# Simulation of a gas blasted liquid sheet on GPU architecture

**CONFERENCE PAPER** · SEPTEMBER 2014

READS

41

**2 AUTHORS**, INCLUDING:

R. Banerjee

Indian Institute of Technology Hyderabad

**28** PUBLICATIONS **88** CITATIONS

SEE PROFILE

# Simulation of a gas blasted liquid sheet on GPU architecture

Rajesh Reddy*[1], Raja Banerjee[1]
[1]Department of Mechanical and Aerospace Engineering,
Indian Institute of Technology Hyderabad, India
*Corresponding author: me10p006@iith.ac.in

**Abstract**
Atomization of liquid fuel from injector in gas turbine combustors is commonly achieved by means of coflowing gas stream. A study of the representative example of this scenario is presented in this paper. An in-house finite volume method based code is developed for this purpose. Volume of fluid (VOF) method is used for capturing the interface. Geometric multigrid method is employed for solving the pressure poisson equation and it is parallelized on Graphics processing unit (GPU) architecture to meet the computational demand required. The solver is validated for standard benchmark test cases. The solver is applied to study the effect of gas to liquid velocity ratio on primary atomization of liquid sheet.

**Introduction**
A wide variety of natural and industrial process contains two or more immiscible fluids which encounter large deformations of fluid interface. Common applications which are of interest to scientific community are gas turbines, fuel injection in automobiles, spray painting etc. Usually these processes are characterized by liquid jet breakup, atomization and subsequently spray formation. Different approaches are followed to achieve efficient atomization in technical devices [1]. Two strategies are often observed: First one is injection of high velocity liquid in to a near quiescent gaseous environment. Other is injection of low velocity liquid with a coflowing gaseous stream of high velocity. Usually the first approach is referred as pressure injection and the second one is referred as air assisted/blast atomization. In both the cases, the flow is expected to be highly unsteady and turbulent. Air assisted atomization is commonly encountered in gas turbine combustors.
Performance of a combustion system can be improved by understanding the fuel atomization process. This understanding requires the investigation of liquid jet in the near nozzle region (primary atomization zone). Spray is very dense in the primary atomization zone and is difficult to characterize by experimentation. Detailed numerical simulation is a powerful tool to study the primary atomization zone. Interface capturing methods used for the simulation should allow for discrete representation of the interface and also the sharpness of the interface has to be maintained throughout the computation. Several approaches are available in the literature to predict interface motion. Volume of Fluid (VOF) [2], Front tracking [3] and Level-Set (LS) [4] methods are among the most commonly used strategies. Fuster et al [5] have presented different simulations relating to primary atomization using VOF method. Recently Chenadec and Pitsch [6] reported numerical simulations of jet breakup resulting from a fully turbulent cylindrical pipe, using VOF method for varying density ratios. Desjardins et al [7] have simulated air assisted breakup of both planar and coaxial liquid layers. Desjardins et al have used level set based methodology for capturing the interface. Raessi and Pitsch [8] have reported breakup of liquid ethanol sheet in presence of shear air.
A simplified example which represents the phenomena of air blast atomization is taken as the subject of the present study.
VOF method proposed by Aulisa et al [9] is chosen for interface capturing in the current work because of its excellent mass conservation properties and ability to handle large interface distortions. Solving these problems numerically requires large computational sources. Traditionally computational demand is met by multi core Message Passing Interface (MPI) communication. Instead of MPI, the present solver relied on GPU architecture for parallelisation. GPU programming is done using Compute Unified Device Architecture (CUDA) by NVIDIA Corporation [10]. There are many CFD applications both in compressible and incompressible flows, which makes use of GPU computing. Solving pressure poisson equation is the most time consuming part in the incompressible multiphase flow solver. A common approach is to accelerate this part of the solver on GPU architecture. Griebel et al [11], Kuo et al [12] and Kelly [13] have applied GPU computing in their own style to study multiphase flows. Subsequent section presents a brief introduction to GPU computing.

**Introduction to GPU computing**

In GPU computing, a Graphics Processor Unit is used in conjunction with CPU. It has become a recent trend in high performance computing to use GPUs for executing a part of the program in parallel. The reason being its high processing power and relatively low cost. Current GPUs are incorporated with hundreds of lightweight cores which can accelerate compute intensive applications substantially. The GPUs are efficient for data parallel applications like Computational Fluid Dynamics (CFD), as it has a Single Instruction Multiple Data (SIMD) device architecture.

**CUDA programming environment**

CUDA Application Programming Interface (API) is a programming model provided by NVIDIA for easy implementation on GPUs. CUDA-C contains set of extensions to C programming language and some CUDA accelerated libraries. CUDA also supports C++ and FORTRAN languages. In GPU terminology the term host refers to CPU and the term device refers to GPU. In general a C program, which is executed by the host calls the GPU to outsource the computational routines via a function called kernel. Kernels are designated by __global__ function qualifier, which specifies a function that is called by host and executed on device.

In CUDA computational grid is organised in to number of blocks, which are equal in size and each block consists of number of threads. The blocks and grids can be organised in to one, two or three dimensions. The programmer has to accordingly map every thread to the actual data structure. Each block in the grid is provided with unique identity and is refererred by built in variables *blockIdx.x, blockIdx.y, blockIdx.z*. Similarly each thread in a block is given a unique identity and is identified using builtin variables *threadIdx.x, threadIdx.y, threadIdx.z*. The size of grid (specified in terms of blocks) and the size of *block* (specified in terms of threads) is passed on to the device via kernel call.

Tesla C2075 GPU model is used for the present computations. GPU architecture, CUDA memory and optimization issues are not presented here in interest of brevity. The reader is referred to CUDA C programming guide [14] by NVIDIA corporation.

**Governing equations**

The present work considers two-dimensional, incompressible, variable density, isothermal flow of immiscible fluids. The mass and momentum conservation equations can be expressed in vector form as

$$\nabla . \mathrm{v} = 0 \tag{1}$$

$$\rho\left(\frac{\partial \mathrm{v}}{\partial t} + \mathrm{v}.\nabla \mathrm{v}\right) = -\nabla p + \nabla .[\mu(\nabla \mathrm{v} + \nabla \mathrm{v}^T)] + \mathrm{g} + F \tag{2}$$

Single set of governing equations (Equation 1 & Equation 2) hold throughout the computational domain. $F$ represents body force term which includes surface tension force per unit volume and gravity, if present. Surface tension term is modelled by Continuum Surface Force (CSF) model by Brackbill et al [15]. The location of the interface is determined by using VOF methodology. VOF methods capture the interface using a marker function known as volume fraction $C$, which is defined as the fraction of the reference fluid occupied in a cell. As each fluid particle conserves its own identity, the volume fraction $C$ is a property moving with the flow and its material derivative should be zero.

$$\frac{\partial C}{\partial t} + \mathrm{v}.\nabla C = 0 \tag{3}$$

From the estimation of volume fraction, the effective density and viscosity in a cell can be obtained as (assuming gaseous phase as reference phase)

$$\rho = \rho_g C + \rho_f (1 - C) \tag{4}$$

$$\mu = \mu_g C + \mu_f (1 - C) \tag{5}$$

The suffix f and g represent the liquid and gas phases respectively.

**Numerical methodology**

Finite volume method is used to solve the flow governing equations. Two-dimensional Cartesian, uniform and collocated grid is used. VOF method basically consists of two steps namely interface reconstruction and its advection. Interface reconstruction is done by Youngs method [16]. For advection, operator split scheme with Eulerian implicit-Lagrangian

explicit method proposed by Aulisa et al [9] is used. This advection scheme has the property of mass conservation up to machine accuracy. Navier-Stokes equations are solved by using a projection algorithm namely Simplified Marker and Cell (SMAC). First order explicit forward in time discretization is used for time derivative. Convective term is discretized using a second order ENO scheme [17]. Space derivatives are discretized using a second order central scheme. To avoid pressure velocity decoupling on collocated grid, momentum interpolation scheme given by Rhie and Chow [18] is used. Pressure poisson equation is discretized with a second order central difference scheme, which results in linear system of equations. In the present solver the system of equations are iteratively solved by Geometric multigrid algorithm. Initially serial version (for single cpu core) of the multigrid algorithm was developed and then it was parallelised on the GPU architecture to accelerate the computations. The following section briefly discusses the current implementation of multigrid algorithm on GPU architecture in CUDA environment.

**Pressure poisson equation on GPU**

The pressure poisson equation in the flow solver is ported on to GPU to accelerate the computation. As the remaining part of the solver is processed on CPU, the required data is transferred to the device memory from the host memory before the GPU execution starts. To achieve high performance, this data transfer between the host and device memory is to be minimised.

V-cycle geometric multigrid is implemented in the present solver. The basic steps in the solution algorithm can be termed as smoothing, restriction and prolongation. The computational grid in CUDA is handled by dividing it into two dimensional blocks. Each block consists, say m x n number of threads. A thread with a unique thread id is created corresponding to every cell center node in the actual computational domain. The residual vector and solution vector are stored sequentially in a 1-D array (double). To optimize the algorithm, matrix coefficients are recomputed in every iteration instead of storing in an array. This is because GPU invests more transistors to arithmetic operations rather than to control flow and memory operations. Iterations are computed from the data in the global memory. The density values required at cell face are obtained by taking harmonic mean of corresponding cell centre values. The current multigrid algorithm uses Gauss-Siedel (GS) method as smoother.

A single kernel is designed for the smoothening step. The grid dimension and block dimension can be specified to the kernel depending on the level of multigrid. In the original GS method, a new value at a node is computed from four connected neighbourhood nodes during each iteration. This task is sequential in nature, as it depends on the sequence of execution of the nodes. For the algorithm to run on parallel threads, it is necessary that there are no dependencies among the variables on different threads. To resolve the issue related to inter-thread dependencies, the current implementation employs Red-Black Gauss-Siedel method [19]. For a second order stencil used to solve 2-D poisson equation, two colours (say red and black) are required to generate sets of points that are not related with each other. Now each colour can be processed separately and the calculations within a color are done in parallel. Each colour is processed sequentially. The threads are mapped to points of one colour at a time. Figure 1 shows coloured domain and thread mapping arrangement for a sample domain size of 8X4 interior cells.

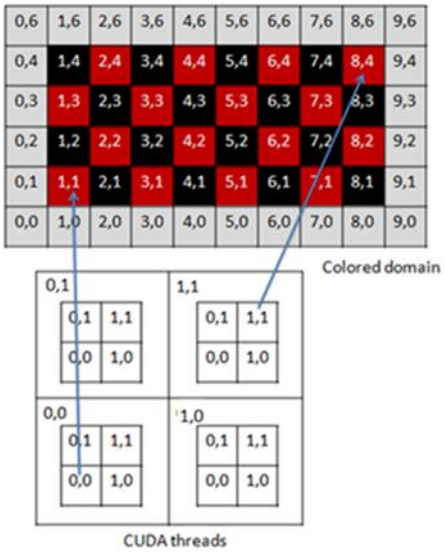



**Figure1.** Colouring of the domain with CUDA thread mapping arrangement.

In restriction operation, coarse grid at any level is chosen by reducing the finer grid size by a factor of two in each direction. So for maximum efficiency of the algorithm, grid sizes of form $2^p$ x $2^q$ is preferred, where p and q are any positive integers. Residual value at a node is obtained by weighted average of eight neighbouring cells. Since the order of execution of nodes does not matter here, restriction operation is convenient to parallelize. A single kernel is used to apply the boundary conditions on all boundaries. For invoking boundary conditions a one dimensional grid is assumed and as many threads as boundary points are created. The one dimensional grid for updating boundaries is divided in to 1-D blocks with each block consisting of 16 threads. It is made sure that threads in one block do not diverge. For the summation of elements in a vector, a 1-D grid is created and is divided into 1-D blocks. Sum of all the elements in a block is calculated by using shared memory, which is several order faster than global memory. Calculations on GPU are done with double precision floating point values.

**Validation of the VOF method**

The performance of the implemented VOF algorithm is tested against a standard test case called as Vortex in a box test introduced by Rider and Kothe [20]. This test suits well to study the combined performance of reconstruction and advection algorithms.

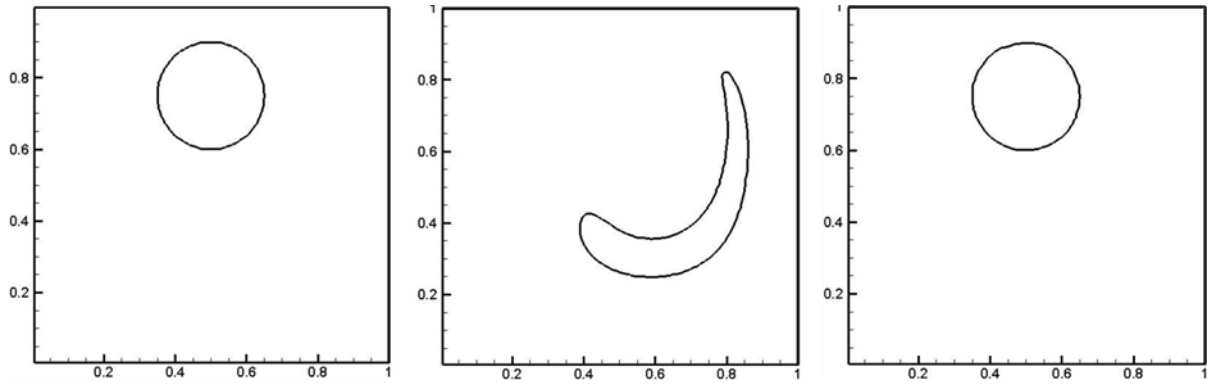
The test problem considers a unit box as computational domain with two fluid phases present. A circular fluid body (phase 1) with radius 0.15 is located with centre (0.5, 0.75) and the rest of the domain is filled with other fluid (phase 2). The applied velocity field is determined by the stream function

$$\psi(x,y,t) = \frac{1}{\pi}\cos(\frac{\pi t}{T})\, sin^2(\pi x)\, sin^2(\pi y) \tag{6}$$

This solenoidal flow field results in a single vortex that will spin fluid elements and stretching them in to a filament that spirals toward vortex center. Due to the temporal cosine term in Equation 6, the fluid has to return to its initial configuration after one cycle. The computations are performed on orthogonal and uniform mesh of size 128 x 128. Two test cases are chosen based on the time period $T$. First case with $T$=2 and second case with $T$=8. Initial CFL number equal to 0.1 is employed for both the cases. Figure 2 shows the interface position at half and full time for $T$ =2 test case. It can be observed that after the deformation of the interface by the flow, it has regained its original shape after one complete cycle. This proves the strength of the VOF scheme employed. Figure 3 shows the interface positions for $T$ =8 test case. Geometrical error $E$ is quantified by using the definition

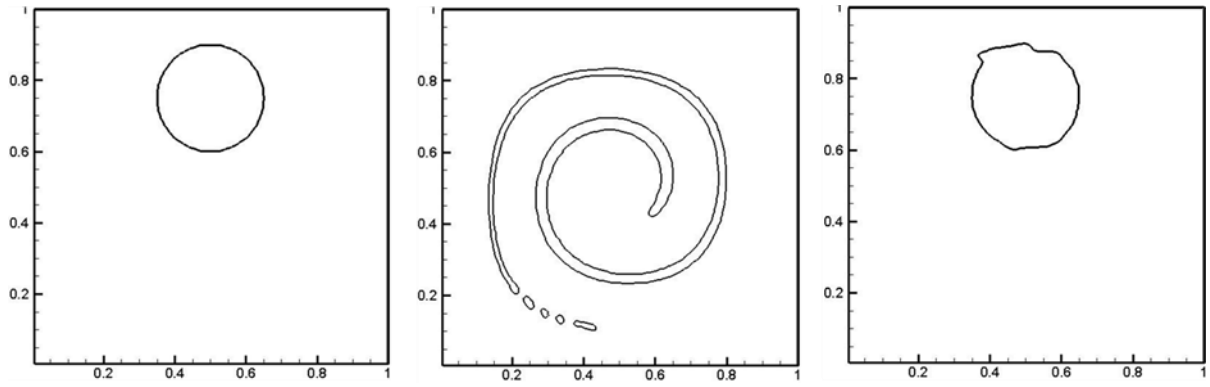
$$E = \sum_{i,j} h^2 \left| C^f_{i,j} - C^i_{i,j} \right| \tag{7}$$

where $h$ is the side of the cell, superscripts $f$ and $i$ refers to final and initial positions respectively. Error is quantified in Table 1.




t=0 t= $T$/2 t= $T$

**Figure 2.** Interface deformation for time period $T$ = 2 case.



t=0 t= $T$/2 t= $T$

**Figure 3.** Interface deformation for time period $T$ = 8 case.

**Table 1.** Error in vortex in a box test.

| Test case | Error |
|---|---|
| $T$ = 2 | 1.08 e-3 |
| $T$ = 8 | 3.56 e-3 |

**Validation of the code**

The developed multiphase flow solver is validated for bubble rise test case presented by Hysing et al [21]. Initial configuration and boundary conditions are illustrated in Figure 4.

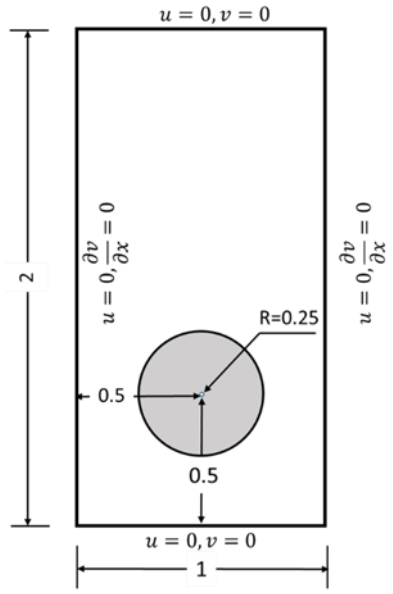



**Figure 4.** Initial configuration and boundary conditions for bubble rise test case.

Physical parameters defining the test case are presented in Table 2.

**Table 2.** Physical parameters used in bubble rise test case.

| Phase | P | μ | σ |
|---|---|---|---|
| Liquid | 1000 | 10 | 24.5 |
| Gas | 100 | 1 | |

The gravitational acceleration $g$ employed in the test case was 0.98. Reynolds number and Eotvos number for the defined test case are 35 and 10 respectively. Characteristic length scale $L$ is equal to $2R$. Characteristic velocity scale $U_g$ is taken as $\sqrt{2gR}$. Time scale is defined here as $t = \frac{L}{U_g}$.

The serial code simulations were performed on a workstation with Intel Xeon 2.8GHz processor and compiled on Linux Redhat C++ compiler (g++). The parallel code simulations are performed on the same workstation with NVIDIA Tesla C2075 GPU with nvcc compiler.

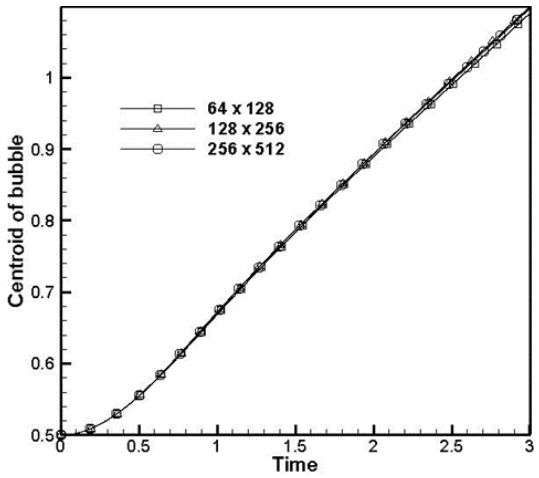



**Figure 5.** Effect of grid resolution on centroid of rising bubble.

Simulations are done on three different grids of sizes 64x128, 128x256 and 256x512. Here grids are chosen to be of the size $2^m$x$2^n$ to gain maximum benefit from the multigrid algorithm employed (m and n are integers). To examine grid independence, the centroid of the rising bubble is plotted against time for the three grid resolutions. From the Figure 5 it can be observed that the three curves are almost indistinguishable. Figure 6 shows the interface shape of the rising bubble at different instants.

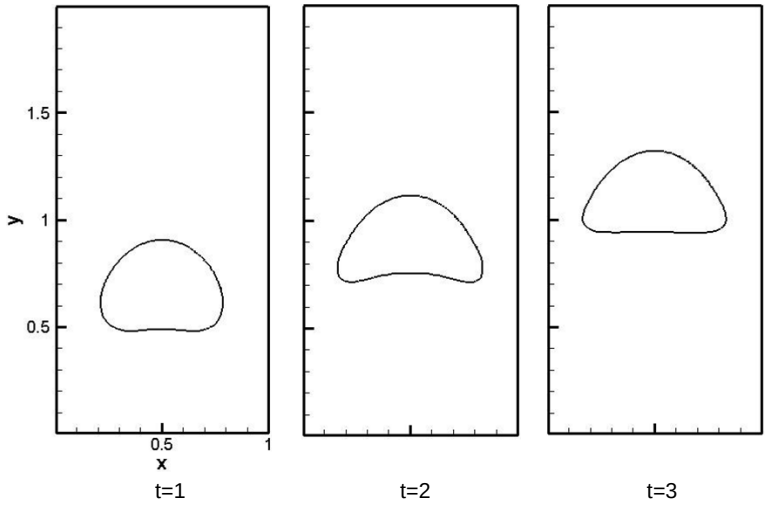
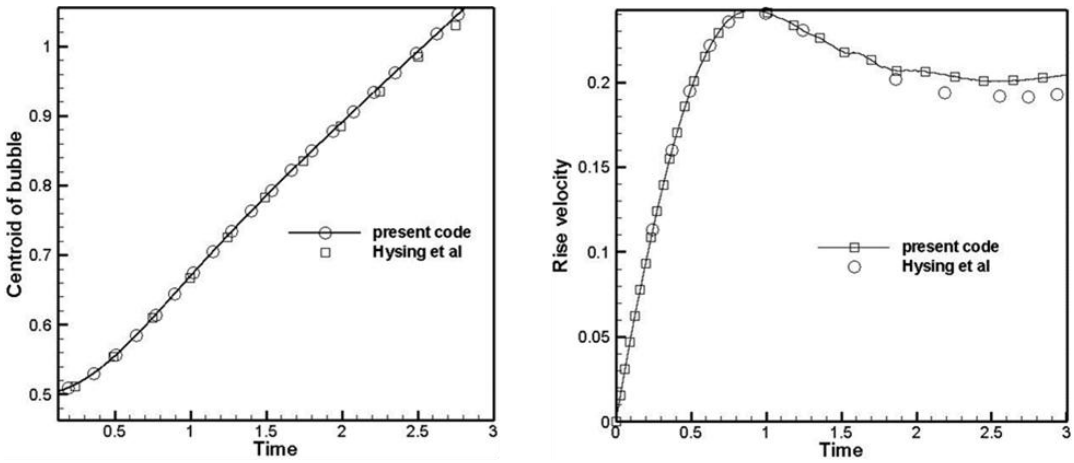



**Figure 6.** Bubble shape at different instants (grid 128 x256).

Here the solution on128x256 grid with time step size equal to $0.5\text{x}10^{-4}$ is compared with the grid independent solution of FreeLIFE group in Hysing et al [21]. Figure 7(a) and (b) shows the comparison of bubble centroid and mean rise velocity respectively. Buoyancy force causes the bubble to rise and the drag force opposes this motion. When all the forces acting on the bubble are balanced, rise velocity shall be constant reaching a steady velocity called as terminal velocity.




(a) (b)

**Figure 7.** (a) Y-coordinate of bubble center with time. (b) Average Y-velocity component of bubble with time. (on 128x256 grid)

Figure 6 and Figure 7 show that the solver has captured the bubble motion and also the results agree satisfactorily with the reference results. Figure 7(b) shows a slight discrepancy in the rise velocity in comparison with Hysing et al. This difference is believed to be resulted from the modelling of surface tension term. Present solver uses standard CSF algorithm by Brackbill et al [15]. The curvature calculation from volume fraction and the resulting jump condition at the liquid/gas interface results in spurious velocity at the interface which becomes increasingly apparent for surface tension dominated problems like the present case. As expected, surface tension effects are strong enough to hold the bubble together and no break up occurs. The bubble has ended in ellipsoidal regime.

**Performance acceleration with the GPU based solver**

The bubble rise test case presented above is simulated both on a single core cpu solver and gpu based parallelized solver, in order to evaluate the performance acceleration. Performance is presented in Table 3 in terms of workunits, which is defined as time taken per control volume per iteration as given by the expression below:

$$workunit = \frac{T_k}{N \times I_k}$$

where $N$ is the number of control volumes in the domain, $T_k$ is the time taken by the pressure poisson solver for $k$ number of time steps and the total number of iterations done during the $k$ time steps is $I_k$.

**Table 3.** Performance of multigrid solver on gpu (Tested by running for first 30 time steps)

| Grid | Workunits | | Speed up |
|---|---|---|---|
| | cpu (xe-7) | gpu (xe-7) | |
| 64x128 | 3.975 | 2.718 | 1.46 |
| 128x256 | 4.092 | 1.373 | 2.98 |
| 256x512 | 4.113 | 0.89 | 4.62 |

**Gas blasted liquid sheet**

The primary atomization of a liquid jet under the influence of co-flowing gas stream is studied. A simplified example which represents the scenario of air blast atomization is presented. Computational domain with boundary conditions is shown in Figure 8. Flow inside the nozzle is not considered here. The present case considers both liquid to gas density ratio and viscosity ratio of 10. This limitation is because the present mathematical formulation shall not hold good at high density ratios. Parameters used in the study are listed in Table 4 (all quantities are in SI units). Three different test cases are studied, which are defined based on the relative velocity between the liquid and gas phase inlet velocities. Gas to liquid velocity ratio of 2.5x, 3x and 4x are considered. Table 5 presents three different cases along with their

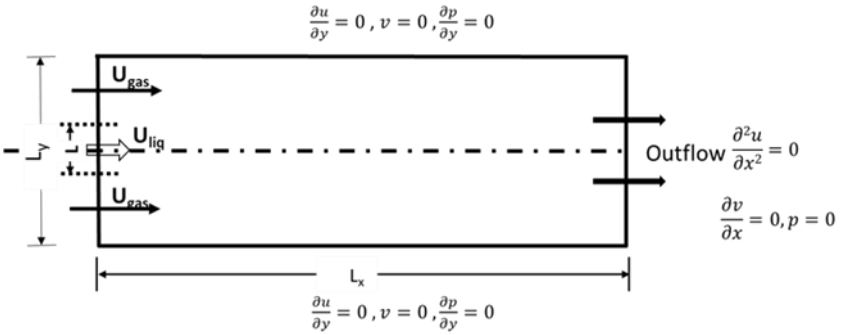



**Figure 8.** Computational domain with boundary conditions.

non-dimensional numbers. The parameters chosen corresponds to Reynolds number and Weber number of the order of $10^3$. Relative velocity between gas and liquid phases at inlet is taken as characteristic velocity and liquid sheet thickness L is the characteristic length. Because of high Reynolds number and weber number, the jet is expected to disintegrate soon after injection.

**Table 4.** Parameters used.

| Phase | ρ | μ | σ | Jet thickness |
|---|---|---|---|---|
| Liquid | 100 | 1e-4 | 0.003 | 1e-4 |
| Gas | 10 | 1e-5 | | |

**Table 5.** Dimensionless numbers for different cases.

| Case | Liquid velocity (m/s) | Gas velocity (m/s) | Reynolds number | Weber number |
|---|---|---|---|---|
| A | 15 | 37.5 | 2250 | 1688 |
| B | 15 | 45 | 3000 | 3000 |
| C | 15 | 60 | 4500 | 6750 |

High Weber number is considered so that surface tension does not dominates the flow characteristics. The computational domain is 18L x 6L.

**Effect of grid resolution on spray morphology**

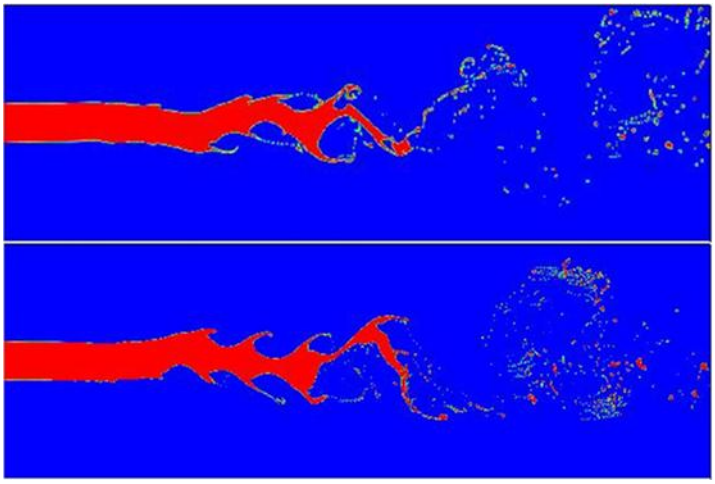


**Figure 9.** Spray formation at an instant of t=60 μs for case B- top (grid 1024 x 256), bottom (grid2048 x 512)

As mentioned by Fuster et al [5], for high Reynolds numbers it is believed that the flow structures in the vortices interact with the small droplets which are already generated ,making the simulations extremely sensitive to mesh size. To check the effect of grid resolution, simulation of case B is performed on two grids of sizes 1024 x 256 and 2048 x 512. Effect of grid resolution on spray morphology is clearly depicted in Figure 9. It is apparent from Figure 9 that coarse grid and fine grid results differ in the oscillation trend, wavelength of instabilities on surface of sheet and amplitude growth. Also, the ligament formation and atomized droplets vary significantly from coarse mesh to fine mesh.
All the simulations for the study are presented on grid size 2048 x 512. For all the three cases presented, no inlet perturbations are imposed.

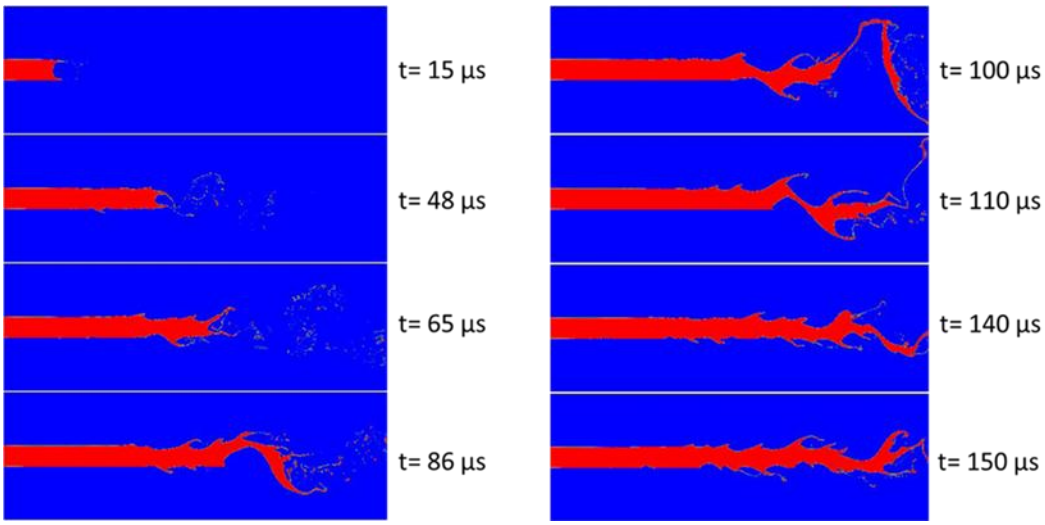



**Figure 10.** Evolution of spray for case A.

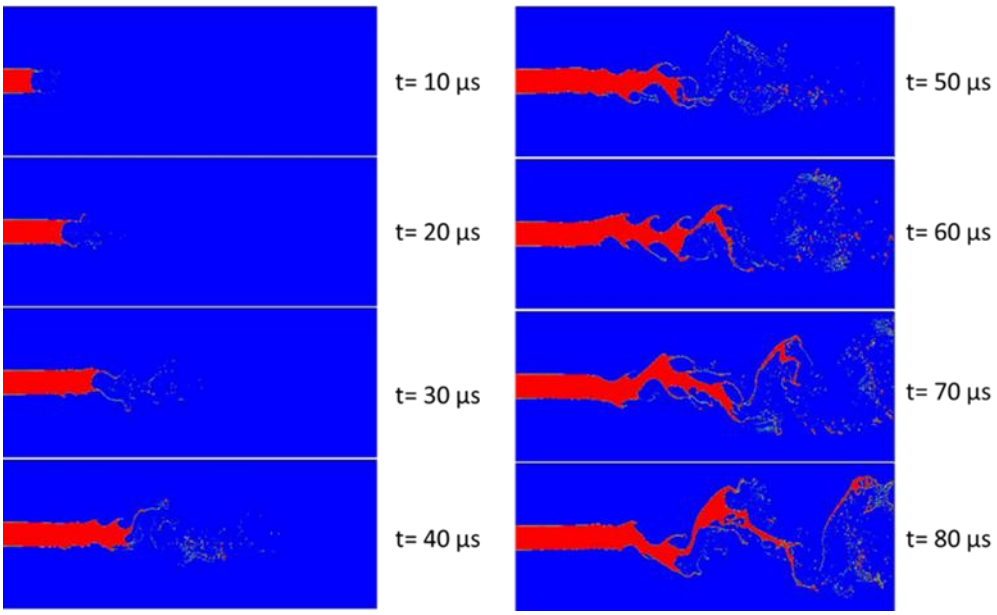



**Figure 11.** Evolution of spray for case B.

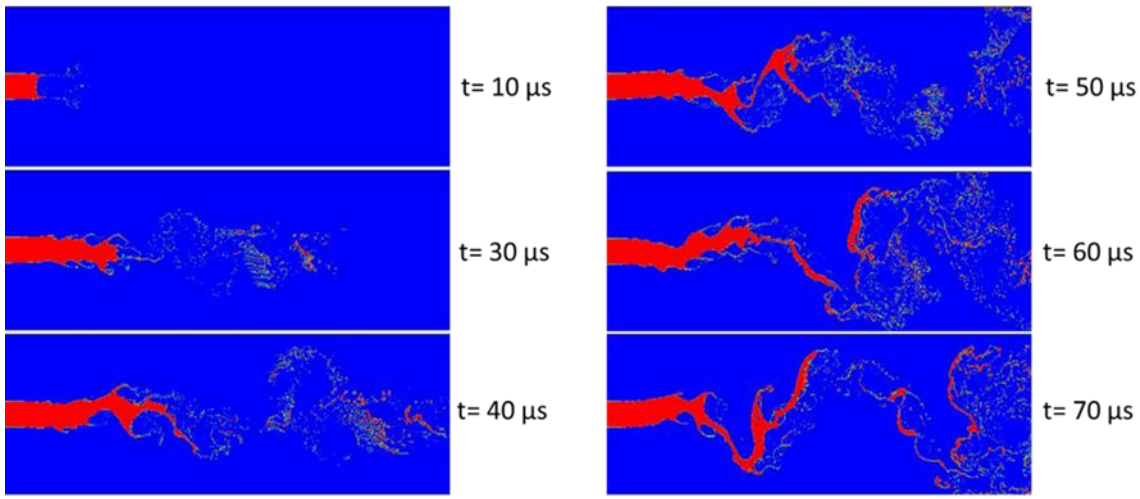



**Figure 12.** Evolution of spray for case C.

Instabilities sets on the surface of the liquid sheet because of the shearing action due to relative velocity between liquid sheet and co-flowing gas stream. This is commonly referred as Kelvin-Helmholtz instability. Figures 10 – 12 show the evolution of the spray for cases A, B and C respectively. The liquid surface is stretched due to the more rapidly moving co-flowing air stream. This results in the formation of thin rims at the tip of the sheet. These rims may then lead to formation of droplets. Pinching of minute droplets from the tip of the jet is observed in the very early stages of injection for all the cases. Also, ligaments start to appear on the surface of the sheet simultaneously which may either collapse on surface of liquid jet or get pinched from the main jet resulting in smaller structures and droplets. Liquid sheet tends to have oscillatory motion as it propagates downstream. The simulations have shown an increase in amplitude of oscillations with the increase in relative velocity between liquid and gas streams. Also, at high Weber numbers planar liquid sheet tends to be more unstable, as it can be seen that the amplitude growth of instabilities is much higher for case C than cases A and B. Rigorous pinching of droplets from ligaments is observed in case C, showing that the momentum created by coflowing stream has a very high impact on liquid sheet atomization.

**Conclusion**

VOF based multiphase flow solver is developed on GPU architecture. Approximate DNS simulations are performed based on the validated solver to study the breakup a planar liquid sheet aided by a coflowing gas stream. To meet the computational demand the present solver relies on GPU based parallelization. Qualitative results depicting the sheet disintegration are presented. Wavy oscillations of the sheet are observed as the flow propagates downstream. It is observed that with the increase in relative velocity between liquid sheet and coflowing gas stream, disintegration of liquid sheet occurs more closely to the injection region. Ligament formation and small droplets pinched off from the main stream were captured because of the fine mesh employed.

**Nomenclature**

| | |
|---|---|
| $u$ | X-component of Velocity [m $s^{-1}$] |
| $v$ | Y-component of velocity [m $s^{-1}$] |
| ρ | Density [Kg $m^{-3}$] |
| μ | Dynamic viscosity [Kg $m^{-1}$ $s^{-1}$] |
| $C$ | Volume fraction |
| $p$ | Pressure [N $m^{-2}$] |
| σ | Surface tension [N $m^{-1}$] |
| $R$ | Radius of bubble |
| L | Liquid sheet thickness |
| $L_x$ | Length in X-direction |
| $L_y$ | Length in Y-direction |
| $U_{liq}$ | Liquid inlet velocity [m $s^{-1}$] |
| $U_{gas}$ | Gas inlet velocity [m $s^{-1}$] |
| t | Time |

**Acknowledgement**
One of the authors (Rajesh Reddy) gratefully acknowledges Dr. Satoshi Ii for his helpful discussions. The authors would like to thank NVIDIA Corporation for hardware support.

**References**

[1] Lefebvre, A. H., 1989, "Atomization and Sprays."
[2] Pilliod, J. E., Puckett, E. G., 2004, Journal of Computational Physics, 199, pp. 465–502.
[3] Tryggvason, G., Bunner, B., Esmaeeli, A., Juric, D., Al-Rawahi, N., Tauber, W., et al., 2001, Journal of computational Physics, 169, pp. 708–759.
[4] Sussman, M., Smereka, P., Osher, S., 1994, Journal of Computational Physics, 114, pp. 146–59.
[5] Fuster, D., Bagué, A., Boeck, T., Moyne, L. L., Leboissetier, A., Popinet, S., et al., 2009, International Journal of Multiphase Flow, 35, pp. 550–65.
[6] Chenadec, V. L., and Pitsch, H., 2013, Atomization and Sprays, 23, pp. 1139–1165.
[7] Desjardins, O., Mccaslin, J.O., Owkes, M., Brady, P., 2013, Atomization and Sprays, 23, pp.1001–1048.
[8] Raessi, M., and Pitsch, H., 2012, Computers and Fluids, 63, pp. 70–81.
[9] Aulisa, E., Manservisi, S., Scardovelli, R., Zaleski, S., 2003, Journal of Computational Physics, 192, pp. 355–364.
[10] Version 2.3., 2009, "NVIDIA CUDA Programming Guide."
[11] Griebel, M., and Zaspel, P., 2010, Computer Science - Research and Development, 25, pp. 65–73.
[12] Kuo, S., Chiu, P., Lin, R., Lin, Y., 2011, World Academy of Science, Engineering and Technology, 52, pp. 878–886.
[13] Kelley, J., 2009, International Mechanical Engineering Congress and Exposition, pp. 2221–2228.
[14] 2012,"NVIDIA Cuda C Programming Guide."
[15] Brackbill, J. U., Kothe, D. B., Zemach, C. A., 1992, Journal of Computational Physics,100, pp. 335–354.
[16] Tryggvason, G., Scardovelli, R., Zaleski, S., 2011. "Direct Numerical Simulations of Gas- Liquid Multiphase Flows".
[17] Chang, Y. C., Hou, T. Y., Merriman, B., Osher, S., 1996, Journal of Computational Physics, 124, pp. 449–464.
[18] Rhie, C. M., and Chow, W. L., 1983, AIAA Journal, 21, pp. 1525–1532.
[19] Demmel, J. W., 1997, “Applied numerical linear algebra."
[20] Rider, W. J., and Kothe, D. B., 1998, Journal of Computational Physics, 141, pp. 112–152.
[21] Hysing, S., Turek, S., Kuzmin, D., Parolini, N., Burman, E., Ganesan, S., et al., 2009, International Journal for Numerical Methods in Fluids, 60, pp.1259–1288.