EXTRACTION OF DROPLET GENEALOGIES FROM HIGH-FIDELITY ATOMIZATION SIMULATIONS

by

Roland Francis Clark Rubel IV

A thesis submitted in partial fulfillment of the requirements for the degree

of

Master of Science

in

Mechanical Engineering

MONTANA STATE UNIVERSITY
Bozeman, Montana

July 2019
DEDICATION

I dedicate this to my parents Roland Rubel III and Robin Vert-Rubel for providing the support and encouragement I received from them throughout this process. As well as my friends and family that helped me accomplish this work.
This material is based upon work supported by the National Science Foundation under Grant No. 1749779. Computational efforts were performed on the Hyalite High-Performance Computing System, operated and supported by University Information Technology Research Cyberinfrastructure at Montana State University.
# TABLE OF CONTENTS

1. INTRODUCTION ..............................................................................................1

2. METHODS ........................................................................................................8

   CFD Solver .................................................................................................8
   Droplet Ancestry Extraction Tool.................................................................9
      Merge Event Identification ......................................................................10
      Split Event Identification ......................................................................11
      Structure Identification ........................................................................12
   Liquid Identification ..................................................................................15
   Structure And Liquid Identification Number Validation ....................17
   Structure Identification Issues .................................................................19
   Parallelization ............................................................................................23
   Extracted Data Into Database .................................................................25
   neo4j Database .........................................................................................26
   MATLAB Post Processing .........................................................................29

3. RESULTS .........................................................................................................31

   Three Dimensional Diesel Jet Simulation ..............................................31

4. CONCLUSION .................................................................................................41

5. FUTURE WORK ............................................................................................43

REFERENCES CITED ..........................................................................................45

APPENDICES ....................................................................................................49

   APPENDIX A : Droplet Genealogy Extraction Code...............................50
   APPENDIX B : neo4j Graphical Database Code .....................................64
   APPENDIX C : MATLAB Post processing Code .....................................67
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Non-Dimensional numbers used for the three dimensional air blast atomizer to test the affect of the interval between time steps for running the droplet genealogy extraction code</td>
<td>21</td>
</tr>
<tr>
<td>2.2</td>
<td>Example of the local list on each processor</td>
<td>24</td>
</tr>
<tr>
<td>2.3</td>
<td>Example of local list combined into the global list before duplicate pairs of $S$ and $L$ have been accounted for</td>
<td>24</td>
</tr>
<tr>
<td>2.4</td>
<td>Example of global list after duplicate pairs of $S$ and $L$ have been accounted for and removed</td>
<td>25</td>
</tr>
<tr>
<td>3.1</td>
<td>Non-Dimensional numbers for a diesel-type jet used to test the droplet ancestry extraction tool</td>
<td>32</td>
</tr>
</tbody>
</table>


LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Image of the four main atomization regimes (a) Rayleigh break up, (b) First-wind induced regime, (c) Second-wind induced regime, and (d) Atomization regime. Reproduced with permission [27].</td>
</tr>
<tr>
<td>1.2</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Image of the jet breakup low-fidelity model method. Reproduced with permission [27].</td>
</tr>
<tr>
<td>1.3</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Image of the sheet breakup low-fidelity model. Reproduced with permission [20].</td>
</tr>
<tr>
<td>2.1</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Corresponding $S$ and $L$ for a merge event.</td>
</tr>
<tr>
<td>2.2</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>Corresponding $S$ and $L$ for a merge event after it has be updated.</td>
</tr>
<tr>
<td>2.3</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Corresponding $S$ and $L$ for a split event</td>
</tr>
<tr>
<td>2.4</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>Corresponding $S$ and $L$ for a split event after being updated.</td>
</tr>
<tr>
<td>2.5</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>Representation of structure identification code stepping through cells and assigning the structure identification number.</td>
</tr>
<tr>
<td>2.6</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>Representation of two droplets moving into a single cell.</td>
</tr>
<tr>
<td>2.7</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>Illustration of the two dimensional multiphase air blast atomization jet configuration. High speed air is injected around a slow moving liquid.</td>
</tr>
<tr>
<td>2.8</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>Structure identification number $S$ at the given time step for the two dimensional air blast simulation.</td>
</tr>
<tr>
<td>2.9</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>Liquid identification number $L$ at the given time step for the two dimensional air blast simulation.</td>
</tr>
<tr>
<td>2.10</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Representation of the repeated splitting and merging events from a droplet moving away from another structure.</td>
</tr>
<tr>
<td>2.11</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>Final time step of the three dimensional air blast atomizer for the 10 time step interval. The different colored droplets represent the different $L$ assigned to the droplets.</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
</tr>
<tr>
<td>----------</td>
<td>------</td>
</tr>
<tr>
<td>2.12</td>
<td>23</td>
</tr>
<tr>
<td>3.1</td>
<td>33</td>
</tr>
<tr>
<td>3.2</td>
<td>33</td>
</tr>
<tr>
<td>3.3</td>
<td>35</td>
</tr>
<tr>
<td>3.4</td>
<td>36</td>
</tr>
<tr>
<td>3.5</td>
<td>36</td>
</tr>
<tr>
<td>3.6</td>
<td>37</td>
</tr>
<tr>
<td>3.7</td>
<td>38</td>
</tr>
<tr>
<td>3.8</td>
<td>39</td>
</tr>
<tr>
<td>3.9</td>
<td>40</td>
</tr>
</tbody>
</table>

2.12 Showing the change in total droplets created by varying the droplet genealogy extraction code time step interval.

3.1 Liquid identification number $\mathcal{L}$ represented by different colors at the final time step in the simulation.

3.2 First 7,500 of the 107,000 droplets formed in the simulation shown in neo4j’s graphical database.

3.3 Average diameter of final droplets based on number of split events.

3.4 Histogram showing the total number of final droplets for the given amount of split events to reach the droplets final state.

3.5 The change in diameter for each split in the final droplets genealogy with three total split events.

3.6 The change in diameter for each split in the final droplets genealogy with four total split events.

3.7 The change in diameter for each split in the final droplets genealogy with five total split events.

3.8 The change in diameter for each split in the final droplets genealogy with six total split events.

3.9 Example of the final droplet’s $\mathcal{L}=12550$ genealogy path shown in Fig. 3.5.
## LIST OF ALGORITHMS

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Creation of the neo4j database from the CSV file</td>
<td>26</td>
</tr>
<tr>
<td>created by the droplet genealogy extraction code</td>
<td></td>
</tr>
<tr>
<td>2.2 MATLAB final droplet extraction from neo4j</td>
<td>30</td>
</tr>
</tbody>
</table>
NOMENCLATURE

\( \rho \phi \) Density
\( \mathbf{u}_\phi = [u, v, w]_\phi \) Velocity field vector
\( t \) Time
\( p_\phi \) Hydrodynamic pressure
\( \mu_{phi} \) Dynamic viscosity
\( g \) Gravitational acceleration
\( \phi \) Phase indicator with values of either \( g \) (gas) or \( l \) (liquid)
\( \Gamma \) Gas-liquid interface
\( S \) or SID Structure identification number
\( L \) or LID Liquid identification number
\( U_{jet} \) Liquid jet velocity
\( U_{cof} \) Co-flow of air velocity
\( D_{jet} \) Jet diameter
\( L_x \) Domain length in \( x \)
\( L_y \) Domain length in \( y \)
\( L_z \) Domain length in \( z \)
\( \sigma_l \) Liquid surface tension
\( D \) Droplet diameter
\( \text{We} \) Weber number
Many research groups are performing high-fidelity simulations of atomizing jets that are taking advantage of the continually increasing computational resources and advances in numerical methods. These high-fidelity simulations produce extremely large data-sets characterizing the flow and giving the ability to gather a better understanding of atomization. One of the main challenges with these data sets is their large size, which requires developing tools to extract relevant physics from them. The main goal of this project is to create a physics extraction technique to compute the genealogy of atomization. This information will characterize the process of the coherent liquid core breaking into droplets and ligaments which may proceed to break up further. This event information will be combined with detailed information such as droplet size, shape, flow field characteristics, etc. The extracted information will be stored in a database, allowing the information to be readily and quickly queried to assist in the development and testing of low-fidelity atomization models that agree with the physics predicted by high-fidelity simulations.
INTRODUCTION

Atomization of liquid jets has many, wide-ranging applications, fuel injection, aerosols, spray painting, and cooling sprays are just a few examples. Even with atomization frequently occurring in everyday life, there is still a lot to learn about this subject. With continually improving computational resources and numerical methods for multiphase flows, extremely accurate high-fidelity simulations have become feasible for many relevant flows. Allowing atomization to be studied in greater detail.

Figure 1.1: Image of the four main atomization regimes (a) Rayleigh break up, (b) First-wind induced regime, (c) Second-wind induced regime, and (d) Atomization regime. Reproduced with permission [27].
Atomization is the process in which droplets break off of a coherent liquid core. Within atomization there are four main regimes the Rayleigh, First-wind induced, Second-wind induced, and Atomization regime that are shown in Fig. 1.1. These regimes are separated by the droplet size relative to the core diameter as well as the core’s length. In the Rayleigh regime the droplets formed are larger than the diameter of the core and occur many nozzle diameters down stream. This is shown in image (a). For the First-wind induced regime the droplets are the same order of magnitude as the core diameter and occur many nozzle diameters down stream similar to image (b). The Second-wind induced regime has droplets that are an order of magnitude smaller than the core diameter and the break up process occurs within a couple nozzle diameters down stream of the nozzle exit as shown in image (c). The last regime as shown in image (d) is the Atomization regime where the droplets are several orders of magnitude smaller than the core diameter and breakup occurs near the nozzle exit. The Reynolds number, the ratio between the inertial forces and the viscous forces, increases as the jet moves from the Rayleigh regime to the Atomization regime. The Weber number, the ratio between the inertial forces and the surface tension forces, range also increases with the transition from the Rayleigh regime to the atomization regime. In the Rayleigh regime the Weber number is around 10, and the Weber number increases to around 35 for the First-wind induced regime. Moving up to the Second-wind induced regime the Weber number increases to around 90, and for the Atomization regime the Weber number increases to over 450.

Most of the modern fuel injectors used in cars are well into the Atomization regime, thus allowing engines to run more efficiently. However, the farther that fuel injectors get into the atomization regime the smaller and more densely packed the droplets become. Therefore, it is increasingly difficult to accurately visualize what is occurring in the atomization process with experimental results.
Due to the difficulties visualizing experimental results, and the expenses in designing and building prototype fuel injectors computational methods are becoming more popular. Computer modeling of fuel injectors is often used by both industry and academia to design injectors and understand atomization. Within computer modeling, there are two main types of atomization models: low- and high-fidelity. Low-fidelity models are most commonly used by industry due to the lower computational cost and run times. Whereas high-fidelity models are more commonly used by researchers that will deal with the longer run times and computational expenses for the increased accuracy.

Computational models such as Reynolds Average Navier-Stokes (RANS) and Large Eddy Simulations (LES) are coupled with low-fidelity atomization models to predict breakup of liquid jets. These models include, for example, Eulerian-Lagrangian Spray and Atomization (ELSA) model [32], Unified Spray Breakup (USB) model [6], Enhanced Taylor analogy breakup (TAB) [29], bag-type breakup (BTB) model [31], multimode breakup (MMB) [30]. These models decrease the computational cost by solving simplified or an assumed form of the Navier-Stokes equations. Since the models are based on assumptions and simplifications they need to be validated against experimental results. Trying to validate these models with simulations that are deep into the atomization regime is becoming increasingly difficult, due to the large number of opaque droplets blocking optical access to the breakup process. Currently, low-fidelity models are forced to compare against experimental results at a point far enough down stream that the spray is diluted enough to be easily visualized or a single droplet is examined. Some examples of this are with the bag-type breakup (BTB) model [31], where they validated against experimental results from Chou and Faeth [5]. Chou and Faeth used single- and double-pulsed shadowgraphy and holography methods to collect droplet data. But
they reported an error (with 95% confidence) of less than 10% for droplet diameters and less than 15% for droplet velocities. For more information on their experimental setup, refer to the works [14], [15], and [16]. Multimode breakup (MMB) [30] was also validated against experimental results from Chou and Feath [5]. The improved TAB model [26] was validated against experimental results found by Krzeckowski [17]. Where Krzeckowski injected single droplets into a wind tunnel and would photograph the process at several different times. Allowing the measurement of droplet diameters from the pictures. Difficulties with these methods start to arise when the type of atomization process being simulated gets farther into the atomization regime causing the droplets to be more densely populated and making it difficult to examine what is actually occurring with experimental results. Krzeckowski’s method [17] works well for validating single droplet models since they can easily visualize those results. However, using this method to validate a model with densely packed droplets does not take into account the affects of the surrounding droplets.

Low fidelity models also change their governing equations based on the type of nozzle design being modeled. The jet breakup model shown in Fig. 1.2 induces a small perturbation on the surface of the jet, which grows until the force is large enough to break small droplets off the core. As compared to Fig. 1.3 where a liquid sheet is injected with sinuous wave perturbations that grow to the point of breaking off large ligaments, that then breakup to form small droplets.

High-fidelity models resolve the moving and deforming gas-liquid interface. Many methods to track the interface have been developed including level set [21] and volume-of-fluid [13] methods and continue to see development. These simulations are extremely computational expensive, taking anywhere from days to weeks to run and generate hundreds of terabytes of data. This is due to the fact that high-fidelity simulations solve the complete Navier-Stokes equations for each cell in the domain,
and must have a small enough grid resolution to capture the smallest scales. By solving the complete governing equations in each cell the simulations produced by high-fidelity results are very accurate. Along with the increased accuracy the benefit of these high-fidelity simulations is that they are not as dependent on the nozzle shape and parameters as low-fidelity models. High-fidelity models have recently started to become increasingly popular. With the increasing popularity of these models many different geometries and simulation parameters have been modeled and validated against experimental results [25], [7], [11], [3], [18], [19], [1].

Even with the increasing popularity of high-fidelity models and the significant amount of information they have available. A very limited amount of data is being extracted from them. Currently, most of the data that is extracted from high fidelity models is similar to the data extracted in experimental results. This is due to the fact that most of the work done with high-fidelity models so far is with the validation of their accuracy, and this is accomplished by comparing to experimental results.
However, now that a lot of these high-fidelity models have had their jet atomization accuracy validated, the work can shift towards extracting more information from these high-fidelity simulations. That is where the droplet genealogy extraction code comes in.

The droplet genealogy extraction code works along with the high-fidelity model to extract specific information from the large amounts of data created by the high-fidelity models. This extracted information can then be used to find new results within the atomization process that have not been found with experimental results. Thus, with the use of high-fidelity simulations and the droplet genealogy extraction code the users can: 1) extract useful information from the high-fidelity simulations, 2) make that information easily accessible for queries, 3) gain a better understanding of the physics occurring in the atomization process, and 4) aid in the improvement
of low-fidelity models. The droplet genealogy extraction code computes droplet ancestry statistics by identifying when liquid structures break apart or merge together. The information of these splitting and merging events, collected from high fidelity simulations, will allow for statistics of droplet dynamics to be produced, and gain a better understanding of the atomization process. The extracted statistics will allow questions like "given a droplet of a given size and shape in a characterized flow, how long will the droplet last and how will the droplet break up?".

The large amount of liquid structures breaking up and merging together throughout the atomization process will be stored in a neo4j graphical database. Graphical databases are used to store large data sets connected through relationships (split and merge events). With this data storage method the information can be easily queried to gain a better understanding of the atomization process in general as well as assisting in the improvement of low-fidelity models. Having the ability to go back in time and probe anywhere inside of the jet gives the ability to see into the densely packed regions of the atomization process that are difficult to visualize experimentally. Allowing low-fidelity models to validate against droplet statistics at any location or time as compared to only validating against data far enough downstream the spray becomes dilute and can be measured. Ideally giving low-fidelity models increased accuracy throughout the entire atomization process.
METHODS

The goal of this work is to create a droplet ancestry extraction tool for high-fidelity atomization simulations and store the extracted information in an easily accessible format. While the high-fidelity atomization simulation is running, the droplet ancestry extraction tool is building a CSV file of all the selected information. Once the CSV file has been completed it can be imported into the neo4j database, allowing the information to be easily queried. Therefore, giving the ability to easily calculate statistical information on the atomization process within the database or with an additional software tool such as MATLAB. This chapter provides details on the high-fidelity CFD solver, the droplet ancestry extraction tool, the neo4j graphical database, and the MATLAB post processing tool.

CFD Solver

The proposed droplet ancestry extraction tool is built into the NGA computational platform [8, 10]. NGA solves conservation laws for mass and momentum in a variable density low-Mach number formulation. The laws for this solver are written as follows:

\[
\frac{\partial \rho_\phi}{\partial t} + \nabla \cdot (\rho_\phi u_\phi) = 0
\]  

(2.1)

and

\[
\frac{\partial \rho_\phi u_\phi}{\partial t} + \nabla \cdot (\rho_\phi u_\phi \otimes u_\phi) = -\nabla p_\phi + \nabla \cdot (\mu_\phi [\nabla u_\phi + \nabla u^T_\phi]) + \rho_\phi g
\]  

(2.2)

where \( \rho_\phi \) is the density, \( u_\phi = [u, v, w]_\phi \) is the velocity field vector, \( t \) is the time, \( p_\phi \) is the hydrodynamic pressure, \( \mu_\phi \) is the dynamic viscosity, and \( g \) is the gravitational acceleration. The subscript \( \phi \) indicates either the gas or liquid phase and takes values
of $\phi = g$ (gas) or $\phi = l$ (liquid). Both of these equations are valid in the gas and liquid phases and are connected by jump conditions at the phase interface. For example, at the interface $\Gamma$ where the jumps of viscosity and density are written as $[\rho]_\Gamma = \rho_l - \rho_g$ and $[\mu]_\Gamma = \mu_l - \mu_g$. If there is no phase change present the velocity field is assumed to be continuous i.e., $[\mu]_\Gamma = 0$. The pressure is discontinuous due to contributions from surface tension and the normal component of the viscous stress, i.e., $[p]_\Gamma = \sigma \kappa + 2[\mu]_\Gamma n^T \cdot \nabla u \cdot n$, where $\sigma$ is the surface tension coefficient and $\kappa$ is the interface curvature.

NGA discretizes the equations using a staggered Cartesian mesh with pressure and other scalars located at the cell centers and velocity components located at cell faces. A second order iterative Crank-Nicolson formulation with a semi-implicit correction on each sub-iteration is used to discretize time [4]. Away from the interface where a smooth solution exists, second-order finite difference operators that are well suited for simulations of turbulence are used for spatial discretization. An unsplit, geometric, semi-Lagrangian method is used near the interface that maintains discrete conservation of mass and momentum [23, 24]. The interface is represented by a geometric volume-of-fluid (VOF) method and the interface curvature is computed with the ACES method [22]. NGA is a highly parallelized code, allowing for simulations to be scaled and decrease run times while being applied to many atomization applications [9, 25, 28].

**Droplet Ancestry Extraction Tool**

The droplet ancestry extraction tool identifies splits and merges during a simulation and stores this information to later be imported into the neo4j graphical database. The splits and merges are identified with two main numbers. The structure identification number $S$, which is a unique number that is assigned to every liquid
structure for every time step. The second number is the liquid identification number \( L \), which is assigned to all the liquid parcels in the domain and moves through time with the liquid parcels. This allows the liquid structures to be tracked through time and gives the ability to find where the liquid structures came from (their genealogy). A merge occurs when multiple \( L \) are found within a single structure with only one \( S \). Similarly a split occurs when multiple structures with different \( S \) are found with the same \( L \). A more detailed explanation of this process can be found below.

Merge Event Identification

A merge event occurs when there is a structure (single \( S \) value) that has multiple \( L \) associated with it. Which can be seen in Fig. 2.1 where the top two droplets merge together to form the bottom droplet. The bottom droplet is assigned the \( S \) of 1 while having the \( L \) of 1 and 2 from the top droplets inside of it. The first step in identifying the merge event is done by sorting the lists by \( S \) and searching the list for duplicate \( S \) values. Which by construction of the list each \( S \) in the list has a
different \( L \) associated with it. With the example shown in Fig. 2.1 searching the list would give two \( S \) of 1. One of the \( S \) would have a corresponding \( L \) of 1 and the other would have a corresponding \( L \) of 2. When a merge event is identified the \( L \) of the new droplet is updated to the \( L \) associated with the structure that has the largest volume of all the structures that participated in the merge event. This can be seen in Fig. 2.2 where the \( L \) of 2 has been updated to 1 because the volume of the \( L \) equal to 2 is less then the volume of the \( L \) equal to 1. Once, all the merge events have been identified for the current time step and the updated list of IDs and volumes have been created. The lists are written to a CSV file to later be imported into the neo4j graphical database.

![Diagram](image.png)

Figure 2.2: Corresponding \( S \) and \( L \) for a merge event after it has be updated.

**Split Event Identification**

The identification of split events is similar to merge events except the lists are sorted by \( L \) instead of \( S \). When there are multiple structures (multiple \( S \)) that have the same \( L \) a split event has occurred. This can be seen in Fig. 2.3 where the top droplet splits off in to the bottom two droplets. The bottom two droplets both have a \( L \) of 1 and \( S \) of 1 and 2. The first step in split identification is sorting the lists by \( L \) and searching for duplicate \( L \) values which by construction means they have different \( S \). For the example in Fig. 2.3 two \( L \) of 1 would be found with a corresponding \( S \) of 1 and a \( S \) of 2. Once, a split event is identified the structure with the largest volume
is found and the $L$ of that structure is left alone. The remaining smaller structures require a new $L$, this is assigned by the next largest integer available, as shown in Fig. 2.4 where the larger droplet with the $S$ of 1 is left alone, and the smaller droplet with the $S$ of 2 has had the $L$ updated from 1 to 2. When all of the splits have been identified for the current time step the list are written to a CSV file to later be imported into the neo4j graphical database. This process of identifying splits and merges is run every 10 time steps to allow the droplets enough time to sufficiently separate. More information on why 10 time steps was chosen is shown the structure identification issues section. Future work will be done to increase the accuracy of the structure identification code so the identification of events can occur on every time step, and do not continuously split and merge back together at every time step until they have separated more then a couple cell widths apart.

Figure 2.3: Corresponding $S$ and $L$ for a split event

Structure Identification

The structure identifier $S$ is a unique integer associated with each liquid structure within the flow domain. The structures are identified using an algorithm proposed by
Herrmann [12]. This algorithm starts by initially going through the domain looking for a cell with liquid inside of it. Once, a cell with liquid is found it is marked with an x. This can be seen in the 1st iteration of Fig. 2.5 where the x denotes the initial cell found with liquid. After this cell has been marked the neighboring cells are inspected for liquid. If the neighboring cells are found to have liquid present they a marked with a n as shown in the 1st iteration Fig. 2.5. Once, this has been completed for the first cell all of the cells marked with an x are then assigned the first available $S$. Then all of the cells marked as n are changed to x, and those cells neighbors are checked for liquid. This can be seen in the 2nd iteration of Fig. 2.5 where the x was set to a $S$ of 1 and the n’s where updated to x’s and the neighboring cells with liquid where marked with n. These steps are repeated until all of the cells withing the structure have been identified as shown by the 3rd and 4th iterations in Fig. 2.5. Once, all of the cells in the initial structure have been identified the process starts to be applied to the next structure. Where the first cell within that structure is marked with an x and the neighboring cells with liquid are marked with a n. This is shown in the 4th iteration of Fig. 2.5. Since, the $S$ value of 1 is already assigned the next available value of 2 is assigned to this structure in the same manner as the first structure as shown in the 5th and 6th iterations of Fig. 2.5.
Figure 2.5: Representation of structure identification code stepping through cells and assigning the structure identification number.
Liquid Identification

The liquid identifier $L$ is a unique integer associated with each liquid parcel in the flow domain. The motion of $L$ is described by the equation

$$\frac{\partial L}{\partial t} + u \cdot \nabla L = 0,$$

(2.3)

where $u$ is the velocity field. This equation is solved with the same geometric semi-Lagrangian scheme used to transport the VOF representation of the phase interface [23, 24]. Using the same discretization of VOF and $L$ transport ensures $L$ moves with liquid parcels even near the phase interface where discontinuities in $L$ exist. Solving the equation geometrically with fluxes at the cell face allows for the identification of merging events. This is done by looking at the fluxes associated with each computational cell and identifying all the $L$ coming into or out of the cell. The semi-Lagrangian method also eliminates numerical diffusion of the $L$ that would occur with a normal discretization method and lead to non-integer $L$ values that do not move consistently with liquid parcels.

Looking at Fig. 2.6 shows two droplets moving into one cell. One droplet has an $L$ of 10 while the other has a $L$ of 5. Solving this with a regular discretization method would cause these values to average when they came within the same cell. The averaging of the $L$ would create a new $L$ of 7.5 (assuming equal mixing of the two droplets in this cell). The value of $L = 7.5$ would cause issues within the droplet genealogy extraction code since it is only set up to work with integer values for $L$, and it would causes issues with the identification of splitting and merging events by creating $L$ that should not have been created. By solving the transport equation with the semi-Lagrangian scheme all of the $L$ values coming into that cell can be taken into account, and one of the current values of $L$ can be used for the merge and transport
of \( \mathcal{L} \) through time. Therefore, no new \( \mathcal{L} \) are being created to change the splitting and merging events. Currently the droplet genealogy code is set up to handle a max of 3 different \( \mathcal{L} \) coming into one cell face at a time. The ability to handle a larger number of \( \mathcal{L} \) can easily be implemented within the code by increasing the allocated space. However, to help save on memory cost only 3 are taken into account currently with a warning message that will be displayed in the case that more then 3 \( \mathcal{L} \) try to merge into one cell face at a time. So far this has not been seen in any simulations run to date. When multiple \( \mathcal{L} \) come into one cell the droplet genealogy code is run even if it is not on its 10th time step interval to account for this merging event. This is done so no merging event information is lost in between the time step intervals for the droplet genealogy extraction code. Currently the first \( \mathcal{L} \) found in this special event is used for the updating and transporting of the \( \mathcal{L} \) to later be used for the split and merge identification.
Structure And Liquid Identification Number Validation

A two dimensional air blast atomizing jet was created for ease and efficiency of testing the proposed droplet ancestry extraction tool. The jets configuration is an air-blast atomizer consisting of slow moving liquid core injected into a high velocity co-flow of air shown in Fig. 2.7. This test case is not physical due to being two dimensional, however this case does quickly create realistic droplet break up events that the droplet ancestry extraction tool can identify.

Figure 2.7: Illustration of the two dimensional multiphase air blast atomization jet configuration. High speed air is injected around a slow moving liquid.

For validation of the droplet ancestry extraction tool, several test cases were run with random sinusoidal perturbations induced on the vertical velocity to generate different break up patterns. Figure 2.8 shows nine different structures in the domain at the given time step. The droplet ancestry extraction tool correctly identifies the nine structures, as indicated by the numbers next to the structures in Fig. 2.8, which are the unique structure identification numbers $S$ assigned by the droplet ancestry extraction tool. Each structure also has its own unique liquid identification number $L$ that is being carried with the liquid parcels. This can be seen in Fig. 2.9 by the
different colored structures that represent the unique $L$. The actual given value for the $L$ is shown next to the structures in Fig. 2.9. The transportation of the $L$ through time can be seen by looking at Fig. 2.9. The first structures to break off end up near the right side of the domain. These structures keep the lower $L$ as they move away from the core. The structures that broke off at a later period in time have been assigned a larger $L$. This shows that the droplet ancestry extraction tool is correctly identifying and updating the $S$ and $L$ for all the structures present in the simulation.

Figure 2.8: Structure identification number $S$ at the given time step for the two dimensional air blast simulation.

Figure 2.9: Liquid identification number $L$ at the given time step for the two dimensional air blast simulation.
Structure Identification Issues

Issues arises with this method when droplets initially split off and have not moved at least several cells away from the core or other droplets. Since the structure identification method looks at neighboring cells and only checks to see if they have a VOF greater then 0. The droplets can get into a situation where they are constantly splitting and re-merging with the core until they have moved several cells away from the core. This can be seen in Fig. 2.10 where a solid liquid core and a single droplet are shown moving through time. At the first time step, the droplet has separated from the core. However, since both neighboring cells have a VOF>0 the structure identification code thinks that these are the same structure. This causes the split event to not actually occur until the second time step. Where their is a cell in between the core and the droplet allowing the structure identification code to identify two separate structures and a split is recorded. At the next time step the core has moved into the next cell and the droplet has not moved into the next cell yet. This causes the two neighboring cells to have a non-zero VOF again and the structure identification code thinks these are one structure and a merge occurs. Finally at the fourth time step, the droplet has moved to the next cell causing the neighboring cells to not all have VOF>0 making the structure identification code identify two separate structures causing a split to be recorded. This process can occur multiple times until the droplet can get far enough away from the core for there to always be a cell with VOF=0 between the structures. Having all of these splitting and merging events occur on droplets that are not actually splitting and merging can skew the final droplet statistics. To deal with this issue the droplet ancestry extraction code is only run every 10 time steps to allow the droplets to sufficiently separate from the core so this constant splitting and merging does not occur.

A three dimensional version of the previous two dimensional air blast atomizer
Figure 2.10: Representation of the repeated splitting and merging events from a droplet moving away from another structure.
was created to quickly test the affects of changing the interval on how often the droplet
genealogy extraction code is run. The non dimensional numbers in Table 2.1 lead to
a mesh size of \( N_x \times N_y \times N_z = 128 \times 64 \times 64 \). These values where chosen to allow the
simulations to be run on a local computer with four cores for fifteen hundred time
steps in 5 hours. Time step delays of 0, 5, 10, 15, and 20 time steps where tested.
Where a time step delay of 0 means that no delay in time steps was taken, and the
droplet genealogy extraction code ran on every time step. A 5 time step delay means
that the droplet genealogy extraction code ran every 5th time step of the simulation.
Fig. 2.11 shows what the jet looked like at the final time step for the 10 time step
delay. Note that, if two droplets enter the same cell (not neighboring) during any
time step, the merger of those droplets is executed independent of the time step delay
to ensure the liquid identification number remain integer values.

<table>
<thead>
<tr>
<th>Number</th>
<th>Definition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reynolds number</td>
<td>( \rho_l U_{jet} D_{jet} / \mu_l )</td>
<td>500</td>
</tr>
<tr>
<td>Weber number</td>
<td>( \rho_l U_{cof} D_{jet} / \sigma_l )</td>
<td>400</td>
</tr>
<tr>
<td>Density ratio</td>
<td>( \rho_l / \rho_g )</td>
<td>40</td>
</tr>
<tr>
<td>Viscosity ratio</td>
<td>( \mu_l / \mu_g )</td>
<td>1.6666</td>
</tr>
<tr>
<td>Velocity ratio</td>
<td>( U_{cof} / U_{jet} )</td>
<td>20</td>
</tr>
<tr>
<td>Domain length</td>
<td>( L_x / D_{jet} )</td>
<td>8</td>
</tr>
<tr>
<td>Domain widths</td>
<td>( L_y / D_{jet} )</td>
<td>4</td>
</tr>
<tr>
<td>Cells across diameter</td>
<td>( D_{jet} / \Delta x )</td>
<td>16</td>
</tr>
<tr>
<td>CFL number</td>
<td>(</td>
<td>u</td>
</tr>
</tbody>
</table>

Table 2.1: Non-Dimensional numbers used for the three dimensional air blast atomizer
to test the affect of the interval between time steps for running the droplet genealogy
extraction code
Figure 2.11: Final time step of the three dimensional air blast atomizer for the 10 time step interval. The different colored droplets represent the different $L$ assigned to the droplets.

The total number of droplets recorded by the extraction tool was used to assess the effect of the time step delay. Figure 2.12 shows how the total number of recorded droplets throughout the simulation quickly drops off as the time step delay starts increasing. The decrease slows until the total number of droplets starts to plateau around the 10 time step delay. Starting out with running the code every time step creates over 3,400 hundred droplets. When the code is run every time step a lot of the droplets that split off from the core end up merging and splitting off from the core over and over again. This is caused from the droplets not actually merging back with the core but they move within one cell of each other so the structure identification code thinks they are part of the same structure even though they have not actually merged together. By increasing the interval at which the droplet genealogy code is run the droplets are allowed time to sufficiently separate so the code does not think they are merging back together. The interval of 10 time steps was chosen since the majority of the back and forth of the splitting and merging with the core has been eliminated. As well as the least amount of information is lost to having large gaps in time between extracting information for the neo4j database.
Future work will be done on the structure identification code to take care of this issue of fictitious splitting and merging. The idea is to look at the droplets and cores normal vectors, and see if the droplets are moving towards the core or away from the core. This will allow the droplets to be identified earlier as well as allow the droplet genealogy extraction code to be run every time step so no information is lost between time steps. However, multiple droplets within the same cell could still not be differentiated and would be considered the same structure.

Parallelization

With NGA being a highly parallelized code the droplet genealogy extraction code needs to be parallelized to work well within NGA. With the procedures mentioned above a value for $S$ and $L$ are provided for each computational cell with liquid parcels present on each local processor. Each local processor then creates a local list of all the $S$, $L$, and volumes. These local lists are then all collected on to the root processor.
with MPI gather where the global lists are created. The global lists are assembled by combining all of the local lists and sorting through them looking for duplicate pairs of $S$ and $L$.

<table>
<thead>
<tr>
<th>$S_{local}$</th>
<th>$L_{local}$</th>
<th>Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Table 2.2: Example of the local list on each processor.

<table>
<thead>
<tr>
<th>$S_{local}$</th>
<th>$L_{local}$</th>
<th>Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.75</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Table 2.3: Example of local list combined into the global list before duplicate pairs of $S$ and $L$ have been accounted for.

Table 2.2 shows an example of two local lists. These list are combined together to form the global list shown in Table 2.3. Once the local lists have been combined together, duplicate pairs of identifiers are created when a single structure exists on multiple processors. These duplicate pairs of identifiers are found and used to find the total volume of the structure by adding all of the cell volumes from each processor together. For example, in Table 2.3 the duplicate pair is the two rows that have a $S$ of 1 and a $L$ of 1. These two rows will be combined into a single row with the volumes added together to represent the entire single droplet. Once the duplicate pair has been accounted for and the volume has been updated, the duplicate row is
removed from the global list. This can be seen in Table 2.4 where the duplicate pairs have been combined into a single row with the total volume. After the duplicate pairs are removed and the lists have been updated, a sorting index list is created to assist in the identification of splitting and merging events. The final part of the parallelization is to broadcast the updated liquid identification numbers $L$ from splitting events to all the processors in order to update the $L$ before the next time step is started.

Table 2.4: Example of global list after duplicate pairs of $S$ and $L$ have been accounted for and removed.

<table>
<thead>
<tr>
<th>$S_{global}$</th>
<th>$L_{global}$</th>
<th>Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1.25</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0.2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Extracted Data Into Database

Databases are excellent tools to store massive amounts of data that is easily queried for information and many different types of databases exist such as: relational, object-oriented, network, and graphical databases. Each database stores information in a slightly different way, allowing different forms of data to be easily stored based on how the user wants the data to be related. With graphical databases any type of relationship between nodes that can be drawn on a whiteboard can be made in the database. Thus, giving the ability to create all the different relationships needed between droplets. Because of the ability to create relationships in this manner the neo4j graphical database was chosen for this project.
Once the NGA high-fidelity simulation of an atomizing jet has finished running, and the droplet genealogy extraction code has created the CSV file. A new neo4j project is set up for the atomizing jet simulation. The CSV file has to then be copied into the import folder for the newly created neo4j project. Once the CSV file is in the neo4j database the Cypher command shown in Appendix B is run to create the droplets genealogy within the database. The pseudo code shown in algorithm. 2.1 summaries the main steps in the code provided in Appendix B.

Algorithm 2.1: Creation of the neo4j database from the CSV file created by the droplet genealogy extraction code

1: **Load**: CSV file

2: **Create**: Nodes based on events stored in CSV file

3: **Match**: droplet nodes

4: **Where**: d.Event equals Merge

5: **Delete**: Merge nodes

6: **Match**: n.droplet and d.droplet

7: **Where**: n.Event equals Split and n.OLD_LID equals d.New_LID

8: **Create**: Split relation from d node to n node

9: **Load**: CSV file

10: **Match**: n.droplet and d.droplet


12: **Create**: Merge relation from d node to n node

It should be noted that in order to run this section of code the Enable multi-statement query editor box needs to be checked in the neo4j settings. The first section
of this code loads the specified CSV file from the import folder and creates all of the nodes for the droplets. As well as assigning all of properties for each droplet such as the droplets old and new $S$ and $L$, the split or merge event time, and volume. However, this section of code not only creates all of the nodes for the splitting events it also creates nodes for the merging events based on the format of the CSV file. An additional step of matching all the droplets that have the event type of Merge and deleting them needs to be taken. Afterwards, the next section of code can be run to create the split relationships between droplets. Once, the split relations have been created they appear as the red lines shown in Fig. 3.2. The split relationship is created by looking at only splitting events and where a nodes old $L$ equals another nodes new $L$. After the split relations have been built the merge relations can then be built. The merge relations are then built in a similar manner as the split relations. By comparing the new and old $L$ to see what droplets merged with each other, which are then represented by the green lines shown in Fig. 3.2. All four parts of the code are run at one time with the use of neo4j’s multi-statement setting to initially build the neo4j database, and thus create all the relationships shown in Fig. 3.2. After the code has been initially run, and the database has been set up the code is no longer used for that database. Only queries to extract information from the database are used.

Some simple queries that can be quickly and easily run in neo4j are shown below.

```sql
// diameter of primary atomization droplets
match (n:droplet)
where n.Old_LID = 1 AND n.Event <> 'None'
return (avg(n.Volume)*3/4/pi())^(1/3), count(n)
```
// average diameter of secondary atomization droplets

match (n:droplet)

where n.Old_LID <> 1

return (avg(n.Volume)*3/4/pi())^(.3333333333), count(n)

These queries find the average diameters of the droplets for both primary and secondary atomization. As well as the total number of primary and secondary atomization events that occurred throughout the simulation.

Both of these codes start out by running the match command, which finds all the nodes within the database marked as droplets. After all of the droplet nodes have been identified a where command is run. This where command uses conditional statements to identify if the droplets split from the core or from another droplet. For primary atomization the conditional is if the droplets old liquid ID is one and the event type is not equal to none. This identifies primary atomization because anything that has an old liquid ID of one had to come from the core. The event not equal to none is only used to eliminate to core node from the search, sense the core always has a liquid ID of 1. After the droplets of primary atomization have been found the return command is run, which converts the volume of the droplets to a diameter based on the equivalent volume of a sphere. The count command is also run in the return line to find the total number of primary atomization droplets throughout the simulation.

The where conditions for secondary atomization are just simply any droplet that has split off and does not have an old liquid ID of 1. If the old liquid ID does not equal 1 the droplet had to split off of another droplet putting it in the secondary atomization category.

Another example query traces the path back from a specified droplet to the core.

// Finds the path from specified droplet back to the core
Match (n:droplet {Event: 'None'}),(d:droplet {New_LID: 14}), p =
shortestpath((n)-[:Split*]->(d))
return [d IN nodes(p) | d.Volume], p, length(p)

This query returns the different droplets that split off from the core to form the
specified final droplet. It also gives the volume of every droplet in the path back
to the core, and how many split events took place to get to that point. To start,
this code runs the match command to identify all the droplet nodes in the project.
However, this match command also uses some conditional statements to narrow the
results down to the requested nodes. The first conditional statement finds the n node
with the event equal to none which is the core. The second conditional statement
finds the d node that has the specified new liquid ID value. Once both the core node
and the droplet node have been identified the shortest path command can be run,
which finds the shortest path back to the core from the specified droplet by split
relationships. After the shortest path command has run the return command is run,
which returns all of the droplets connected by split relations between the core and
the specified droplets. The return command also gives all the droplets volumes and
the total number of split relations in that path. Having this information allows for
interesting statistical analysis of droplet genealogy to occur as shown later on in the
results chapter.

MATLAB Post Processing

With the creation of the neo4j database MATLAB can be used for additional post
processing of the data. MATLAB has the ability to connect to the neo4j database
and extract information from it. This is done by giving MATLAB the username,
password, and the localhost address that was created for the neo4j database. After
MATLAB has that information the MATLAB commands in Appendix. C can be run
to extract all the information from the neo4j database for further statistical analysis. The following pseudo code shown in algorithm 2.2 walks through that process.

Algorithm 2.2: MATLAB final droplet extraction from neo4j
1: **Connect:** to neo4j database
2: **Cypher command:** find all droplets with no split or merge events leaving the droplet
3: **For:** 1 to total end droplets found
4: **Cypher command:** find shortest path, volumes, and path length
5: **End:**
6: **Calculate:** diameter cutoff
7: **For:** 1 to total end droplets found
8: **If:** end drop diameter less than or equal to the diameter cutoff save values in final droplets array
9: **End:**

This command finds all of the droplets that do not have any splitting or merging events leaving them. Then loops over all of those final droplets and finds their paths back to the core. While these paths are being found the volumes for each droplet in the paths are stored in a matrix for analysis. The biggest benefit with using MATLAB for some of the post processing of the neo4j data is the ability to use for loops, since for loops do not exist in the neo4j language. However, future work within neo4j is planned to find a way to do most of the queries currently completed within MATLAB in neo4j instead. Due to the amount of time it takes MATLAB to communicate back and forth with the neo4j database and extract information. If all of the queries could be run in neo4j the analysis of the data could be done significantly faster.
RESULTS

In order to show the capability of the droplet ancestry extraction tool and the database created by its extracted information a three dimensional diesel jet simulation was run. Some of the possible statistical information from the simulated jet is calculated and displayed in the following chapter.

Three Dimensional Diesel Jet Simulation

The simulation uses a three dimensional axial diesel-type jet characterized by the non-dimensional numbers in Table. 3.1, which results in a mesh size of $N_x \times N_y \times N_z = 1024 \times 128 \times 128$. The values in Table. 3.1 and the mesh size where chosen to simulate a realistic diesel jet that was capable of running within a week on the Hyalite High-Performance Computing System with 160 processors. With the specified Reynolds number, Weber number, and the size of the mesh the small turbulent scales and gas-liquid interface features will not be resolved within the simulation. However, the goal of this test is just to demonstrate the capabilities of the droplet ancestry extraction tool and not to studying a fully resolved atomizing diesel jet, which is recommended to be done in future work.

Figure 3.1 shows a rendering of the atomized jet at the final time step of the simulation. The gas-liquid interface is shown and colored by the liquid identification number $L$. For all of the droplets, the genealogy extraction code has not only assigned and updated all of the droplets ID’s throughout the simulation, but it has also stored all of that information in the CSV file to be represented in the neo4j database. This is shown in Fig. 3.2 where the blue dots represent the droplets. The red lines represent splitting events and the green lines represent merging events. For each droplet, all the information exported from the droplet genealogy extraction code is stored on the
<table>
<thead>
<tr>
<th>Number</th>
<th>Definition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reynolds number</td>
<td>$\rho_l U_{jet} D_{jet} / \mu_l$</td>
<td>25000</td>
</tr>
<tr>
<td>Weber number</td>
<td>$\rho_l U^2_{jet} D_{jet} / \sigma_l$</td>
<td>10000</td>
</tr>
<tr>
<td>Density ratio</td>
<td>$\rho_l / \rho_g$</td>
<td>40</td>
</tr>
<tr>
<td>Viscosity ratio</td>
<td>$\mu_l / \mu_g$</td>
<td>1.6666</td>
</tr>
<tr>
<td>Domain length</td>
<td>$L_x / D_{jet}$</td>
<td>60</td>
</tr>
<tr>
<td>Domain widths</td>
<td>$L_{y,z} / D_{jet}$</td>
<td>7.5</td>
</tr>
<tr>
<td>Cells across diameter</td>
<td>$D_{jet} / \Delta x$</td>
<td>17.06</td>
</tr>
<tr>
<td>CFL number</td>
<td>$</td>
<td>u</td>
</tr>
</tbody>
</table>

Table 3.1: Non-Dimensional numbers for a diesel-type jet used to test the droplet ancestry extraction tool.

The total amount of stored data is drastically reduced by this process. Even though this simulation is relatively coarse, the simulation generated around 8 TB of data. This amount consists of the visualization data and all properties for every time step. Comparatively, after split and merge events were identified and stored in neo4j the size of the database was 400 MB. Having this decrease in total data storage allows for a wider range of simulations to be stored for low-fidelity modeling validation.

With the data now in the neo4j database, queries can be made to gain a better understanding of the atomization process. For example, looking at the final time step the droplet ancestry extraction code found there to be 29,798 events of primary atomization (droplets splitting from core) with an average diameter of 26.6 $\mu$m. For secondary atomization (droplets splitting from other droplets) a total of 77,362 events where found with an average diameter of 17.1 $\mu$m. This shows that the majority of
the droplets formed in the atomization process split off from other droplets instead of splitting directly off of the core and not breaking up any further.

Figure 3.1: Liquid identification number $\mathcal{L}$ represented by different colors at the final time step in the simulation.

Next, the results were narrowed down by selecting final droplets for further analysis. Final droplets were selected with the criteria that 1) they did not have any splitting or merging events leaving them and 2) they were small. The first criteria is a natural definition of a final droplet and is shown in Fig. 3.2 as any droplet

Figure 3.2: First 7,500 of the 107,000 droplets formed in the simulation shown in neo4j’s graphical database.
that does not have any of the split (red lines) or merges (green lines) leaving the droplet (blue dot). The second part of the criteria ensured the droplets were small. This requirement was imposed so none of the larger droplets that had just broken off of the core near the end of the simulation would be taken into account. These large droplets would continue to break up as the simulation progressed through time. Because of this, these droplets would not be an accurate representation of droplets that have reached their final size and are done breaking up. Therefore, skewing the results of how droplets come to their final size. This criteria was implemented by selecting droplets with a Weber number less than 10, which occurs when the surface tension force is large enough compared to inertial forces to keep the droplet from breaking up [2]. With a Weber number set to be less then 10 the Weber number equation can be solved for a corresponding diameter.

\[
\begin{align*}
\text{We} &< 10 \\
\text{We} &= \frac{\rho_l U_{\text{jet}}^2 D}{\sigma} < 10 \\
D &< \frac{10 \sigma}{\rho_l U_{\text{jet}}^2}
\end{align*}
\]

where \( D \) is the droplet diameter, \( \text{We} \) is the Weber number, and \( U_{\text{jet}} \) is the liquid velocity. The corresponding diameter for the Weber number of 10 was calculated to be \( 5 \times 10^{-7} \) m for this simulation.

Figure 3.3 shows how the the average diameter of the final droplets changes based on the number of splitting events that occurred to create that droplet. For the first splitting event the average diameter of the droplets is fairly large at 53 \( \mu \)m then for every split event the average diameter smoothly decays until around the 7th split event where the droplets start to stay roughly the same size until the 10th split event where all the droplets are below the final droplet size. Before the 10th split event
Figure 3.3: Average diameter of final droplets based on number of split events.

there are larger structures present that are increasing the average of the droplets even though the majority of the droplets are below the diameter cutoff. This graph gives a good idea off roughly what size droplets to expect based on how many splitting events they have previously gone through for low-fidelity model comparison.

Looking at Fig. 3.4 shows how many split events each final droplet took to reach its final size. Most of the droplets took 2 to 4 splitting events to reach their final size. Very few of the final droplets took more then seven splitting events to reach their final size. And the majority of the droplets should be to their final size in under five splitting events.

Figure 3.5 takes a more in depth look at the final droplets and the path they took to get to their final size. All of the droplets that took three splitting events to reach their final size in Fig. 3.4 are shown in Fig. 3.5. After the first split event the majority of droplets are all very similar in size. Then for the next splitting event there is two very distinct paths the droplets take the first path is they have a split event occur that does not change the overall size of the droplet much. Then the droplets
Figure 3.4: Histogram showing the total number of final droplets for the given amount of split events to reach the droplets final state.

Figure 3.5: The change in diameter for each split in the final droplets genealogy with three total split events.
split off to their final droplet size at the next splitting event. For the second path the droplets seem to take a drastic drop in size on the second splitting event compared to the first path. Then on the third splitting event they decrease slightly in size to reach the final droplet size.

Fig. 3.6 shows all the final droplets that took four splitting events to reach their final size shown in the Fig. 3.4 histogram. After the first split most of the droplets have a similar size compared to the droplets that took three splitting events to reach their final size. On the second split event most of the droplets only slightly dropped in size. The second splitting event did not have the distinct bi-modal event occur. The bi-modal splitting event did not occur until the third splitting event. This bi-modal event had very similar changes in droplet size as the droplets with three splitting events. It just occurred at the third splitting event instead of the second.

Fig. 3.7 shows all the droplets that took five splitting events to reach their final size shown in Fig. 3.4. These droplets follow a similar path as the three and four
splitting event cases. The droplets slowly drop in size every split event until the second to last splitting event. Where the bi-modal event occurs and the droplets drop to similar sizes in that event as the previous bi-modal events did. The same similar path is seen in Fig. 3.8 with the droplets that took six splitting events to reach their final size. The droplets slowly decreased in size every split event until the second to last split event where the bi-modal event occurred again.

Figure 3.7: The change in diameter for each split in the final droplets genealogy with five total split events.

Comparing Fig. 3.3 to Figs. 3.5, 3.6, 3.7, and 3.8 shows that there are a couple interesting things occurring within this atomization process to inform low-fidelity models. Fig. 3.3 has a very smooth decay in droplets diameter between splitting events. Compared to the distinct bi-modal decay that the individual droplet decays are showing.

Looking at the bi-modal event closer it seems like all of the larger droplets either split in to small sized droplets that then proceed to split into the final sized droplets. Or they go directly from the large droplet size to the final droplet size in one event.
Figure 3.8: The change in diameter for each split in the final droplets genealogy with six total split events.

Showing how most of the final droplets are all formed by breaking off of a large structure once that larger structure has moved far enough away from the core to not cause the small droplet to reemerge with surrounding larger droplets or the core. This information is showing a distinct path that the droplets are taking to reach their final size, which gives more information for low-fidelity model developers to create models that agree with the physical process of atomization.

In Fig. 3.7 and Fig. 3.8 some events of where the droplets split off to very small sizes early on can be seen. However, these droplets did not stay at this final size they end up merging with larger structures before they start to split off again. These small structures that split off early on while they are still near the core or larger structures mostly seem to end up merging back with the core or a larger structure. Until they have moved far enough away from the core.

Future work will be conducted on how the flow field characteristic affect the break up process. To see if there is a relation between the flow field characteristics
and how the bi-modal break up events are always occurring at the second to last splitting event for the droplets. As well as future work needs to be done on the change of droplet size relative to simulation time and life time of the droplet for more comparison data for low-fidelity models. Giving the ability to look at different simulation times and locations, and be able to know what kind of droplet shapes and sizes should be present as well as how many to expect.

![Figure 3.9: Example of the final droplet’s $L=12550$ genealogy path shown in Fig. 3.5.](image)

One of the final droplets paths that took three splitting events to reach its final size was randomly chosen. The selected final droplet had a $L=12550$ and each split event was found and displayed in Fig. 3.9. At the top of Fig. 3.9 is the first split event as shown in Fig. 3.5. Where the first droplet splits off from the core and is assigned the $L=7800$. Continuing through time until the next splitting event that is shown by the middle jet. The $L=7800$ droplet splits off and creates the $L=9885$ droplet. Until the final split event occurs for that droplet shown by the bottom jet in Fig. 3.9. Where the $L=9885$ droplet splits off to finally form the $L=12550$ droplet. This shows how the droplets are moving apart from each other through time as they split apart as well as where these droplets are coming from originally. This information would be very useful in the development of low-fidelity models.
CONCLUSION

In this work, a droplet genealogy extraction tool was developed to gather useful information from high-fidelity simulations of atomizing sprays. This information was put into a manageable and easily queried form within the neo4j graphical database. In order to help gain a better understanding of the atomization process in general as well as aid in the development of low-fidelity models.

The droplet genealogy extraction code extracts useful information from simulations performed with the NGA computational platform by the use of structure identification and liquid identification numbers. These numbers are used to identify splitting and merging events to be sent to the neo4j database along with the event times, droplet size, structure and liquid identification numbers. Once this information is in neo4j, useful statistics on the way droplets break apart and arrive at their final size can be found.

Two distinct ways that the droplets decay were found. One was the smooth decay of the final droplets average diameters based on splitting events. These droplets quickly decayed in size on the first few splitting events, and then proceeded to decay less and less each splitting event thereafter. Eventually reaching a plateau where the droplet size did not very much. The second was the final droplets bi-modal decay event that always occurred at the second to last splitting event for the droplet. This event of bi-modal decay could be caused by the droplets entering a specific flow field condition resulting in that particular breakup behavior. However, additional research is needed to insure this was the case. The majority of the final droplets took 2 to 4 splitting events to reach their final size and very few droplets took more then 7 splitting events to reach their final size. The difference in the primary atomization droplet size verse the final droplets primary atomization size shows how their are a
lot of initially small droplets coming off the core. But the individual droplet paths show how these small droplets quickly merge back with larger structures until they have reached a point far enough away from the core.

As mentioned earlier there are some issues with the structure identification code that cause inaccurate splitting and merging events to occur. However, by delaying when the droplet genealogy extraction code runs, a significant change in total droplets found was observed. Allowing the delay of ten time steps for the droplet genealogy extraction code to be used in order to eliminate most of the repeated splitting and merging events with the least amount of information lost due to large time gaps. Until future work can be completed to increase the accuracy of the structure identification process.

With this work, a method to extract useful information from the large amounts of data created by high-fidelity simulations was developed. As well as a method to easily query the extracted information for statistical analysis, creating a way to gain a better understanding of the atomization process in general and assist in the development of low-fidelity atomization models.
Several improvements to the work provided in this paper can be made in future work:

1. Improve the structure identification code to include the normal vectors. Giving the ability to accurately decide which direction the droplets are heading, and know if a merge is going to occur or the droplets are moving apart after a split event.

2. Improve the code that creates the merge relation in neo4j to run more efficiently. By eliminating the back and forth communication with the CSV file for every droplet to create the merge relation.

3. Implement more of the statistical analysis currently done in MATLAB within neo4j, in order to reduce the statistical analysis's computational time.

4. Use the currently included event time data for statistical analysis. Showing how the droplet size changes based on time to further aid the development of low-fidelity models.

5. Add the ability to store the droplets surrounding flow field characteristics during the splitting and merging events, thus giving the ability to see how the flow field changes the droplet behavior throughout the atomization process.

6. Add the ability to find the droplets volume at any time in the simulation instead of just the droplets final volume. By storing the change in volume for every split and merge event the droplets partake in, on the split and merge relations within neo4j.
Additional information could also be added to the neo4j database in future work such as:

1. Run multiple simulations with varying Reynolds and Weber numbers to start building up the neo4j database for low-fidelity model validation.

2. Run fully resolved simulations to get more accurate data extraction to increase the accuracy of the extracted droplet statistics.

3. Apply machine learning to the neo4j database to possibly help in development of lower order models.
REFERENCES CITED


APPENDICES
APPENDIX A

DROPLET GENEALOGY EXTRACTION CODE
The following droplet genealogy extraction code is an additional post processing code that has been added to the NGA computational platform. An atomizing diesel jet simulation needs to run in NGA along with this droplet genealogy extraction code to work. This code has been written in the FORTRAN computer language.

```fortran
! Module to track liquid structures and their ancestries!
!
! SID = Structure IDs
! - unique to each structure
!
! LID = Liquid IDs
! - moves with liquid parcels
!
module dump_structID
  use multiphase

  ! Pointer to LsID in scalar
  real(WP), dimension(:,:,:,:), pointer :: LsID

  ! Number of splits and merges
  integer :: eventCount
  ! New LID number
  integer :: LID_count
  ! Counter to know when to run split merge code
  integer :: n_timesteps

end module dump_structID

! ________________!
! Initialization!
! ________________!
subroutine dump_structID_init
  use dump_data
  use dump_structID
  use dump_struct
  implicit none
  integer :: n
  logical :: found_LsID=.false.

  ! Initialize event counter
  eventCount=0

  ! Initialize number of timesteps counter
  n_timesteps = 0
```

! Find scalar LsID
do n=1,nscalar
  if (trim(SC_name(n)).eq.'LsID') then
    found_LsID=.true.
    exit
  end if
end do

if (found_LsID) then
  ! Set pointer to correct scalar
  LsID_nSC=n
  LsID(имино_:имакс_,jмино_:jмакс_,kмино_:kмакс_)=SC(:, :, :, n)
else
  call die('LsID not found in list of scalars')
end if

! Initialize LID Counter
call parallel_max(nint(maxval(LsID)),LID_count)

! Initialize output file
if (irank.eq.iroot) then
  open(1, file="merge_split.csv", form="formatted", status="replace",
    action="write")
  write(1, '(A,A)') 'EventCount,Merge_Split,OldLID,OldSID,NewLID,
    NewSID, ' &
    'Vol,Time,'
  write(1, '(A)') '0,None,1,0,1,0,1,0,'
  close(1)
end if

! Remove LsID from cells without interface
where (.not.hasInt .and.VOFavg.lt.0.5_WP) LsID=0.0_WP
return
end subroutine dump_structID_init

! ___________________
! Update sIDs      !
! ___________________
subroutine dump_structID_update
  use dump_data
  use dump_structID
  use dump_struct
  use quicksort
use memory
use multiphase_fluxes
implicit none

integer :: n,m,i,j,k,nStruct,nRem,ierr
integer, dimension(nproc) :: g_nStruct_array, displace
integer :: largest, nStructNSplit, nStructNMerge
! Variables to deal with multiple LIDs due to fluxes
integer :: nLID, nLID_cell
integer, dimension(10) :: LID_cell
! Lists to identify splits and merges
integer :: g_nStruct, newLID, LIDlargest
integer, dimension(:), allocatable :: SID, LID
real(wp), dimension(:), allocatable :: sVol
! Global lists
integer, dimension(:), allocatable :: g_SID, g_LID, g_IND,
g_tmpInt, sortI
real(wp), dimension(:), allocatable :: g_VOL, g_tmpWP
! Update lists
integer :: nUp
integer, dimension(:), allocatable :: up_SID, up_LIDo, up_LIDn

! Make LsID nearest integer to avoid accumulation of errors
LsID=nint(LsID)

! Initialize arrays
allocate( SID(10) ); SID=huge(0)
allocate( LID(10) ); LID=huge(0)

! Initialize structure count
nStruct=0

! loop over domain to identify structures
do k=kmin_,kmax_
    do j=jmin_,jmax_
        cell : do i=imin_,imax_

        ! Search existing lists for this structure
        if ((id(i,j,k)).ne.0) then

        ! Make list of LsIDs for this cell (self + any fluxes into this cell)
            LID_cell=0
            nLID_cell=0

        end if

        end do

    end do

end do
! x_i flux of LID
do n=1,maxLsID
  if (SLF_LsIDx(i, j, k, n) .gt. epsilon(0.0, WP) .and. 
    all(SLF_LsIDx(i, j, k, n) .ne. LID_cell(1:nLID_cell))) then
    nLID_cell=nLID_cell+1
    LID_cell(nLID_cell)=SLF_LsIDx(i, j, k, n)
  end if
end do
! x_i+1 flux of LID
do n=1,maxLsID
  if (SLF_LsIDx(i+1, j, k, n) .lt. -epsilon(0.0, WP) .and. 
    all(abs(SLF_LsIDx(i+1, j, k, n)) .ne. LID_cell(1:nLID_cell))) then
    nLID_cell=nLID_cell+1
    LID_cell(nLID_cell)=abs(SLF_LsIDx(i+1, j, k, n))
  end if
end do
! y_j flux of LID
do n=1,maxLsID
  if (SLF_LsIDy(i, j, k, n) .gt. epsilon(0.0, WP) .and. 
    all(SLF_LsIDy(i, j, k, n) .ne. LID_cell(1:nLID_cell))) then
    nLID_cell=nLID_cell+1
    LID_cell(nLID_cell)=SLF_LsIDy(i, j, k, n)
  end if
end do
! y_j+1 flux of LID
do n=1,maxLsID
  if (SLF_LsIDy(i, j+1, k, n) .lt. -epsilon(0.0, WP) .and. 
    all(abs(SLF_LsIDy(i, j+1, k, n)) .ne. LID_cell(1:nLID_cell))) then
    nLID_cell=nLID_cell+1
    LID_cell(nLID_cell)=abs(SLF_LsIDy(i, j+1, k, n))
  end if
end do
! z_k flux of LID
do n=1,maxLsID
  if (SLF_LsIDz(i, j, k, n) .gt. epsilon(0.0, WP) .and. 
    all(SLF_LsIDz(i, j, k, n) .ne. LID_cell(1:nLID_cell))) then
    nLID_cell=nLID_cell+1
    LID_cell(nLID_cell)=SLF_LsIDz(i, j, k, n)
  end if
end do
all(SLF.LsIDz(i,j,k,n).ne.LID_cell(1:nLID_cell))) then
  nLID_cell=nLID_cell+1
  LID_cell(nLID_cell)=SLF.LsIDz(i,j,k,n)
end if
end do
! z_{k+1} flux of LID
do n=1,maxLsID
  if (SLF.LsIDz(i,j,k+1,n).lt.epsilon(0.0,W)) &
     all(abs(SLF.LsIDz(i,j,k+1,n)).ne.LID_cell(1:nLID_cell))) then
    nLID_cell=nLID_cell+1
    LID_cell(nLID_cell)=abs(SLF.LsIDz(i,j,k+1,n))
  end if
end do
! LID in cell
if (nint(LsID(i,j,k)).ne.0) then
  if (all(nint(LsID(i,j,k)).ne.LID_cell(1:nLID_cell))) then
    nLID_cell=nLID_cell+1
    LID_cell(nLID_cell)=nint(LsID(i,j,k))
  end if
else
  ! Transport LID into cell using fluxes – choosing first fluxed LsID
  if (nLID_cell.gt.0) then
    ! Flux exists
    LsID(i,j,k)=LID_cell(1)
  else
    print *, 'ID=',id(i,j,k)
    print *, 'Warning: cell with liquid but no LID','
  end if
end if

! Add LIDs/SIDs to lists if
! – merge within a cell (always process) or
! – nSteps is reached
if (nLID_cell.gt.1.or.mod(ntime,10).eq.1) then
  ! Loop over the LIDs associated with this cell
  do nLID=1,nLID_cell
    ! Search lists for match with same SID and LID
    do n=1,nStruct
if ( SID(n) .eq. id(i,j,k) .and. &
     LID(n) .eq. LID_cell(nLID) ) then !
   Found match
   ! Go to next cell/LID_cell
   cycle cell
   end if
end do

! Didn’t find a match in current lists --> Add
   new row
nStruct = nStruct +1
SID(nStruct)=id(i,j,k)
LID(nStruct)=LID_cell(nLID)

! Reallocate arrays if necessary
if ( nStruct .eq. size(SID) ) then
   call reallocate_int_1D(SID,100)
   call reallocate_int_1D(LID,100)
   ! Set initial values
   SID(nStruct+1:size(SID))=huge(0)
   LID(nStruct+1:size(LID))=huge(0)
end if
end if
end do cell
end do
end do

! Compute structure volumes (locally)
allocate(sVol(nStruct))
do n=1,nStruct
   sVol(n)=sum(vol(imin_:imax_,jmin_:jmax_,kmin_:kmax_)*VOFavg( &
               imin_:imax_,jmin_:jmax_,kmin_:kmax_), &
               mask=id(imin_:imax_,jmin_:jmax_,kmin_:kmax_)==SID(n). &
               and.nint(LsID(imin_:imax_,jmin_:jmax_,kmin_:kmax_) ==LID(n))
   end do

! ________________
! Sync lists across processors
! Number of structures on each processor
{
call MPI_AllGather(nStruct,1(MPIINTEGER,g_nStruct_array,1,
   MPIINTEGER,comm,ierr)
g_nStruct=\text{sum}(g_nStruct_array)
}

! Allocate global lists
allocate(g_SID(g_nStruct)); g_SID=0
allocate(g_LID(g_nStruct)); g_LID=0
allocate(g_IND(g_nStruct)); g_IND=0
allocate(g_VOL(g_nStruct)); g_VOL=0.0,WP
allocate(sortI(g_nStruct))

! Compute displacements for data from each processor in global arrays
{
do n=1,nproc
   displace(n)=\text{sum}(g_nStruct_array(1:n-1))
end do
}

! Gather lists onto root
{
call MPI_GATHERV(SID(1:nStruct),nStruct,MPIINTEGER, &
g_SID,g_nStruct_array,displace,MPIINTEGER,iroot =1,comm,ierr)
call MPI_GATHERV(LID(1:nStruct),nStruct,MPIINTEGER, &
g_LID,g_nStruct_array,displace,MPIINTEGER,iroot =1,comm,ierr)
call MPI_GATHERV(sVOL(1:nStruct),nStruct,MPIREAL,WP, &
g_VOL,g_nStruct_array,displace,MPIREAL_WP,iroot =1,comm,ierr)
}

! Root works with global lists
if (irank.eq.iroot) then

   ! Construct index to identify each structure
   g_IND=昐(n,n=1,g_nStruct) /)

   ! Check for duplicates in global lists
   !
   ! Sort by SID
   sortI=昐(n,n=1,g_nStruct) /)
   call quick_sort_dint(g_SID,sortI)
g_LID=g_LID(sortI)
g_VOL=g_VOL(sortI)
g_IND=g_IND(sortI)

end if
! Loop through list and combine structures with the same SID and LID
nRem=0
SIDloop : do n=1,g_nStruct−1
! Update g_nStruct and nRem
g_nStruct=g_nStruct−nRem
nRem=0
! Loop over other structures in list
do m=n+1,g_nStruct−nRem
  ! Check if SID's match
  if ( g_SID(n) .eq. g_SID(m−nRem) ) then
    ! Check if LIDs match
    if ( g_LID(n) .eq. g_LID(m−nRem) ) then ! combine structures
      ! Sum volumes
      g_VOL(n)=g_VOL(n)+g_VOL(m−nRem)
      ! Shift lists
      g_SID(m−nRem: g_nStruct−1)=g_SID(m−nRem+1: g_nStruct)
g_LID(m−nRem: g_nStruct−1)=g_LID(m−nRem+1: g_nStruct)
g_VOL(m−nRem: g_nStruct−1)=g_VOL(m−nRem+1: g_nStruct)
g_IND(m−nRem: g_nStruct−1)=g_IND(m−nRem+1: g_nStruct)
    ! Update counter on number of rows that have been removed
    nRem=nRem+1
  else
    ! Do nothing, different structures
  end if
else ! SID's do not match => move onto next SID
cycle SIDloop
end if
end do
! Update g_nStruct without removed rows
g_nStruct=g_nStruct−nRem

! Remove deleted rows
allocate(g_tmpInt(g_nStruct))
allocate(g_tmpWP(g_nStruct))
g_tmpInt=g_SID(1: g_nStruct); deallocate(g_SID); allocate(g_SID(g_nStruct)); g_SID=g_tmpInt
g_tmpInt = g_LID(1:g_nStruct); deallocate(g_LID); allocate(g_LID(g_nStruct)); g_LID = g_tmpInt
g_tmpWP = g_VOL(1:g_nStruct); deallocate(g_VOL); allocate(g_VOL(g_nStruct)); g_VOL = g_tmpWP
deallocate(g_tmpInt,g_tmpWP)

! Recreate g.IND with combined structures
deallocate(g.IND)
allocate(g.IND(g_nStruct))
g.IND = (/ (n,n=1,g_nStruct) /)

! Allocate update lists
nUp = 0
allocate(up_SID (2*g_nStruct))
allocation(up.LIDo(2*g_nStruct))
allocation(up.LIDn(2*g_nStruct))

!if(n_timesteps .eq. 5) then
  ! SPLIT
  !
  ! Sort by LID list
  sortI = (/ (n,n=1,g_nStruct) /)
call quick_sort_dint(g.LID,sortI)
g.SID = g.SID (sortI)
!g.NewLID = g.NewLID(sortI)
g.VOL = g.VOL (sortI)
g.IND = g.IND (sortI)

! Initialize split counter
nStructNSplit = 1

! Loop over structures looking for split (same LID, diff SID)
do n = 1,g_nStruct-1

  ! If this structure was part of a previous split -> skip
  if (nStructNSplit .gt. 1) then
    nStructNSplit = nStructNSplit - 1
    cycle
  end if

  ! If the next LID equals the current LID split occurred
do \( m = n+1, g.nStruct \)
    if \( (g.LID(n).eq.g.LID(m)) \) then
        nStructNSplit=nStructNSplit+1
    else
        exit
    end if
end do

! Cycle if not a split
if (nStructNSplit.eq.1) cycle

! Identify largest structure
largest=maxloc(g.VOL(n:n+nStructNSplit−1),1)
LIDlargest=g.LID(n+largest−1)
!print *, 'Splitting largest vol ', g.VOL(n+largest−1), ', all vols ', g.VOL(n:n+nStructNSplit−1)

! Loop over structures in this split event to update LIDs
do \( m = 1, nStructNSplit \)
    ! All structures but largest get new LID
    if (m.ne.largest) then
        ! Store old LID in update lists to send to procs
        nUp=nUp+1
        up.SID(nUp)=g.SID(n+m−1)
        up.LIDO(nUp)=g.LID(n+m−1)

        ! Structure needs new LID
        LID_count=LID_count+1
        newLID = LID_count
        g.LID(n+m−1)=newLID

        ! Store new LID in update lists to send to procs
        up.LIDn(nUp)=newLID
    end if
end do

! Update event counter
eventCount=eventCount+1

! Write Split information to file
open(1, file="merge_split.csv", form="formatted", status="old", position="append", action="write")
do  m=1, nStructNSplit  
  if (m.ne. largest) then
    print *, 'SPLITTING', g_LID(n+m-1), 'from', g_LID(n+largest-1), '!! newLID=', g_LID(n+m-1)
    write(1, "(I0 ,A, I0 ,A, I0 ,A, I0 ,A, I0 ,A, ES22.16, A, ES22.16, A)"") eventCount, 'Split', g_LID(n+largest-1), ',', ', g_SID(n+largest-1), ',', &
    g_LID(n+m-1), ',', g_SID(n+m-1), ',', g_VOL(n+m-1), ',', time, ',', '
  end if
end do
close(1)

! MERGES
!
! Sort by SIDlist
sortI=(/ (n,n=1,g_nStruct) /)
call quick_sort_dint(g_SID, sortI)
g_LID = g_LID (sortI)
g>NewLID = gNewLabel(sortI)
g_VOL = g_VOL (sortI)
g_IND = g_IND (sortI)

! Initialize split counter
nStructNMerge=1

! Create list to hold new LID's needed by split routine
allocate (gNewLabel(g_nStruct)); gNewLabel=g_LID
do n=1, g_nStruct -1
  ! If this structure was part of a previous merge -> skip
  if (nStructNMerge.gt.1) then
    nStructNMerge=nStructNMerge-1
    cycle
  end if
  ! If the next SID equals the current SID merge occurred
  do m=n+1, g_nStruct
if (g.SID(n) .eq. g.SID(m)) then
  nStructNMerge=nStructNMerge+1
else
  exit
end if
end do

! Cycle if not a merge
if (nStructNMerge.eq.1) cycle

! Identify largest structure
largest=maxloc(g_VOL(n:n+nStructNMerge−1),1)

! Update structure volume to the combined volume
g_VOL(n:n+nStructNMerge−1)=sum(g_VOL(n:n+nStructNMerge−1))

! New LID of merged structure
newLID=g.LID(n+largest−1)

! Store new LID in update lists to send to procs
do m=1,nStructNMerge
  nUp=nUp+1
  up_SID (nUp)=g_SID(n+m−1)
  up_LIDo(nUp)=g_LID(n+m−1)
  up_LIDn(nUp)=newLID
end do

! Update event counter
eventCount=eventCount+1

! Write merge information to file
open(1, file="merge_split.csv", form="formatted", status="old", position="append", action="write")
do m=1,nStructNMerge
  if (m.ne.largest) then
    print *, 'MERGING', g.LID(n+largest−1), 'and', g.LID(n+m−1)
    write(1, '(10, A, 10, A, IO, A, IO, A, IO, A, ES22.16, A, ES22.16, A)') eventCount, 'Merge', g.LID(n+m−1), g_SID(n+m−1), g.LID(n+largest−1), g_SID(n+largest−1), g_VOL(n+m−1), time
  end if
end do
close(1)
end do

end if ! Root done working on global lists

! Deallocate global lists
deallocate(g_SID,g_LID,g_IND,g_VOL)

! Update LsID using update lists
! Number of updates in lists
call MPI_Bcast(nUp,1,MPI_INTEGER,iroot-1,comm,ierr)

! Communicate update lists to all procs
if (irank.ne.iroot) then
  allocate(up_SID(nUp))
  allocate(up_LIDo(nUp))
  allocate(up_LIDn(nUp))
end if

call MPI_Bcast(up_SID(1:nUp),nUp,MPI_INTEGER,iroot-1,comm,ierr)
call MPI_Bcast(up_LIDo(1:nUp),nUp,MPI_INTEGER,iroot-1,comm,ierr)
call MPI_Bcast(up_LIDn(1:nUp),nUp,MPI_INTEGER,iroot-1,comm,ierr)

! Each proc updates LsID (global array)
do n=1,nUp
   where (id.eq.up_SID(n) .and. nint(LsID).eq.up_LIDo(n)) LsID=
        up_LIDn(n)
end do

! Remove LsID from cells without liquid
where (not.hasInt .and. VOFavg.lt.0.5_WP) LsID=0.0_WP

! Deallocate local lists
deallocate(SID,LID,sVOL,up_SID,up_LIDo,up_LIDn)

return
end subroutine dump_structID_update
APPENDIX B

NEO4J GRAPHICAL DATABASE CODE
This work used the following Cypher code to create the droplets genealogy relationships from the imported CSV file, as well as calculate droplet statistics within the neo4j database.

```cypher
// creates nodes and split/merge relations
load csv with headers from "file:///C: testjet.csv" as csvline

// Deletes merge nodes
match (d:droplet)
where d.Event = 'Merge'
delete d;

// creates split relation
match (n:droplet),(d:droplet)
where n.Event = "Split" and n.Old_LID = d.New_LID
create (d)−[:Split]−(n);

// creates merge relation
load csv with headers from "file:///C: testjet.csv" as csvline
match (n:droplet),(d:droplet)
create (d)−[:Merge]−(n);

// delete all splits and nodes
match ()−[r:Split]−()
delete r;

match ()−[r:Merge]−()
delete r;

match (nn:droplet)
where nn.Event = 'Split' OR nn.Event = 'None'
delete nn;

// max and min diameter of secondary atomization droplets
```
match (n:droplet)
where n.Old_LID <> 1
return (max(n.Volume)*3/4/pi())^(.3333333333), (min(n.Volume)*3/4/pi())^(.3333333333), count(n)

// max and min diameter of primary atomization droplets
match (n:droplet)
where n.Old_LID = 1 AND n.Event <> 'None'
return (max(n.Volume)*3/4/pi())^(.3333333333), (min(n.Volume)*3/4/pi())^(.3333333333), count(n)

// average diameter of secondary atomization droplets
match (n:droplet)
where n.Old_LID <> 1
return (avg(n.Volume)*3/4/pi())^(.3333333333), count(n)

// diameter of primary atomization droplets
match (n:droplet)
where n.Old_LID = 1 AND n.Event <> 'None'
return (avg(n.Volume)*3/4/pi())^(.3333333333), count(n)

// Finds the path from specified droplet back to the core
Match (n:droplet {Event: 'None' }), (d:droplet {New_LID: 14 }), p =
shortestpath((n) −[:Split*]->(d)),
return [d IN nodes(p) | d.Volume], p, length(p)

// Finds final droplets
match(n:droplet) where not (n)−[:Split]->() and not (n)−[:Merge]->()
with collect(n) as nodes
return nodes
APPENDIX C

MATLAB POST PROCESSING CODE
This work used the following MATLAB code to connect to the neo4j database for further post processing of the data.

```matlab
clear all; clc;
username = 'neo4j';
password = 'Full3Djet';

conn = database('','username',password,'neo4j-jdbc-driver-3.3.1.jar','jdbc:neo4j:bolt://localhost:11001');

curs = exec(conn,'match(n:droplet)where not n-[:Split]->() and not n-[:Merge]->() return n.New_LID');
curs = fetch(curs);
enddrops = cell2mat(curs.Data);
volumes = nan(length(enddrops),15);
New_LID = nan(length(enddrops),15);
pathlength = nan(length(enddrops),1);
for i = 1:length(enddrops)
    x = enddrops(i);
    neo4jinput = strcat('Match(n:droplet{Event:"None"})
        (d:
            droplet{New_LID:”,
                num2str(x),”}
        ,p=shortestpath((n)-[:Split*]->(d)) return
            d.INnodes(p) | d.Volume, length(p),
            d.INnodes(p) | d.New_LID ’)
    ’);
    curs = exec(conn,neo4jinput);
    curs = fetch(curs);
    pathlength(i) = curs.Data{1,2};
    volumes(i,1:pathlength(i)+1) = str2num(cell2mat(cell(curs.Data(1,1))));
    New_LID(i,1:pathlength(i)+1) = str2num(cell2mat(cell(curs.Data(1,3))));
end

We = 10;
sigma = 0.005;
Ujet = 10;
rhol = 1000;
Dia_min = We*sigma/rhol/Ujet^2;
Dia = (volumes.*3./4./pi).^((1/3));

Dia_end = nan(2,length(enddrops));
figure(1); clf(1);
for n = 1:length(enddrops)
    j = sum(isnan(Dia(n,:)));
    if Dia(n,end-j)<= Dia_min && Dia(n,end-j) > 0
        Dia_end(2,n) = Dia(n,end-j);
```
Dia_end(1,n) = pathlength(n);

if pathlength(n) == 6
    semilogy(1:pathlength(n),Dia(n,2:pathlength(n)+1))
    hold on
end

drawnow
end

title(’Diameter Of Droplets vs Number Of Splits’)
ylabel(’Diameters(m)’)
xlabel(’Number Of Split Events’)
hold off

figure(2)
histogram(Dia_end(2,:))

figure(3)
histogram(Dia_end(1,:))
title(’Distribution Of Split Events For Final Droplets’)
ylabel(’Number Of Final Drops’)
xlabel(’Number Of Split Events’)

figure(4)
for i = 1:8
    m(i) = mean(Dia(:,i+1), ’omitnan’);
end
plot(m*1e6)
title(’Final Droplets Average Diameter Based On Split Events’)
ylabel(’Diameter(mum)’)
xlabel(’Number Of Split Events’)