Received: 21 June 2019 | Revised: 21 January 2020 | Accepted: 11 March 2020

DOI: 10.1002/fld.4833

RESEARCH ARTICLE

WILEY

The double-tree method: An $O(n)$ unsteady aerodynamic lifting surface method

Bryn Jones | **Peter Dunning** | **Alireza Maheri**

School of Engineering, University of Aberdeen, Aberdeen, Scotland

Correspondence
Bryn Jones, School of Engineering, University of Aberdeen, Aberdeen AB24 3UE, Scotland.
Email: bryn.jones@abdn.ac.uk

Funding information
University of Aberdeen

Summary

Two new methods for reducing the computational cost of the unsteady vortex lattice method are developed. These methods use agglomeration to construct time-saving tree structures by approximating the effect of either a group of vortex rings or query points. A case study shows that combining the two new $O(n \cdot \log n)$ tree methods together results in an $O(n)$ method, called the double-tree method. Other case studies show that the trade-off between accuracy and speed can be easily and reliably controlled by the agglomeration cutoff distance. For a flat plate with 5×200 panels analyzed over 20 time steps, the double-tree method is 7 times faster than the unsteady vortex lattice method with a <5% difference in the force distribution and total lift coefficient. The case studies suggest that the computational benefit will increase for the same level of accuracy if the size of the problem is increased, making the method beneficial for full-aircraft analysis within optimization or dynamic load analysis, where the computational cost of the unsteady vortex lattice method can be large.

KEYWORDS

aerodynamics, incompressible flow, parallelization, subsonic, model, reduction, time integration

1 | INTRODUCTION

The unsteady vortex lattice method (UVLM) is a well-established and understood mid-fidelity aerodynamic analysis tool. One of the most useful descriptions of the method is given by Katz and Plotkin,[1] whom provide a detailed description of each necessary component of the UVLM.

An excellent review of the applications of the UVLM is provided by Murua et al.[2] They state that the UVLM is important in applications with a significant degree of geometric complexity. This can include, for example, rotating blades, such as in wind turbines,[3,4] or large-deflection wings, such as flapping wings,[5,6] morphing wings,[7-9] and highly flexible wings.[10-12] These applications are mainly focused on preliminary design and optimization studies that are made possible when the analysis of geometrically complex systems becomes feasible.

The computational cost of the UVLM is quite large despite its moderate level of fidelity. From the basic UVLM theory, the influence of every panel to every other panel is calculated; furthermore, the total number of panels increases each time step due to the requirement to shed a row of wake panels on the trailing edge of the wing.[1] This means that the order of computations is $O(n^2)$, where n is the number of panels, and that the total number of panels over the course of an

analysis can grow to be large, even for a relatively small wing. On top of this, the analysis may be run many times across optimization loops and/or load cases. This combination of issues leads to a high computational cost. For this reason, significant effort by researchers has been put toward reducing this cost.

There are two possible methods of reducing the computational cost, which are to reduce the size of the problem, or to reduce the required order of computations, both of which can be applied in tandem. Perhaps the simplest method to reduce the size is to truncate the wake after a specified number of time steps, but doing so introduces some error into the solution. Ghommem et al[6] truncate the wake at 10 chord lengths, claiming a "negligible" level of error is introduced. An investigation by Werter et al[13] on the effect of wake truncation on the error introduced into the analysis indicates the lift coefficient error from a 10-chord wake truncation at approximately 0.7%. This drops to 0.2% and 0.05% error in lift coefficient when the wake is truncated at 20 and 40 chord lengths, respectively. However, this error is measured relative to a wake truncation of 80-chord lengths, which is not necessarily as close to the true solution as it is to these other truncated solutions. The size of an analysis truncated at this number of chord lengths is also potentially not small.

The other method of reducing the computational cost is in reducing the required order of computations. Willis et al[14] achieve this with the fast multipole method (FMM), which they applied to an unsteady panel method. This is the main inspiration for the method introduced in this paper and has been applied to the UVLM by Wales et al.[15] The most recent FMM UVLM implementation is by Kebbie-Anthony et al.[16,17] The method requires the conversion of UVLM vortex filaments into vortex particles, at which point the FMM can be implemented with a Lamb-Helmholtz decomposition.[18] The effect of a group (agglomeration) of vortex particles is then estimated using a multipole expansion, allowing for a tree structure to be built that describes the wake in varying levels of agglomeration.

Using the tree structure, any chosen point in the domain can then have its wake influence calculated in terms of arbitrary wake agglomeration around that point.[14] This means that the wake can effectively have full refinement close to the point, and progressively less refinement further away from the point, which saves computation time while minimizing the difference in the solution. Kebbie-Anthony et al show that the UVLM FMM can achieve $O(n)$ computations[17]; however, the procedure is reasonably complex to implement, and its performance is sensitive to several parameter values.

This paper outlines a new tree-based method that does not require the conversion of vortex filaments into vortex particles, nor the implementation of the FMM, and whose performance is sensitive to only a single parameter. Thus, it is simpler to implement and easier to tune the performance. Two new methods of agglomeration based on UVLM theory are presented, each of which gives rise to a new tree-based method. The double-tree method achieves $O(n)$ computations by combining both tree methods together.

The next section will discuss the UVLM method, followed by the filament tree method, the query point tree method, and finally the double-tree method. This is followed by the results section, which shows that each tree method individually is $O(n \cdot \log n)$ time, whereas the double tree is $O(n)$. Extensive investigation into the induced solution difference (compared with the standard UVLM) and computational savings are also provided, along with a discussion of parallelization, followed by conclusions and recommendations for further work.

2 | UVLM METHOD

The UVLM is an incompressible time-domain 3D lifting surface method used to simulate the flow of a fluid around a solid object.[1] The solid object is discretized into a structured grid of aerodynamic panels, aligned with the leading edge of the object, placed at the boundary of the object and the fluid.

Each panel is represented by a combination of a *vortex ring* and a *control point*. The leading edge of the vortex ring is placed at the quarter-chord location of the panel, while its sides are coincident with the panel sides.[1] The trailing edge of the ring is determined by the quarter-chord location of the panel immediately downstream of the current one. The control point is placed in the center, at the three-quarter-chord of the panel. A diagram of this layout is shown in Figure 1.

The vortex ring is comprised of four singular vortex filaments that each share the same strength value. Each vortex filament induces a tangential velocity on the entire flow field proportional to its strength and length, and inversely proportional to the distance from the filament, shown by the equation given by Bertin and Smith for the velocity at a point due to a vortex filament[19]:

$$V = \frac{G}{4\pi} \frac{r_1 \times r_2}{|r_1 \times r_2|^2} \left(r_0 \cdot \left[\frac{r_1}{|r_1|} - \frac{r_2}{|r_2|} \right] \right), \tag{1}$$

where Γ represents the vortex strength, and each vector is defined in Figure 2.

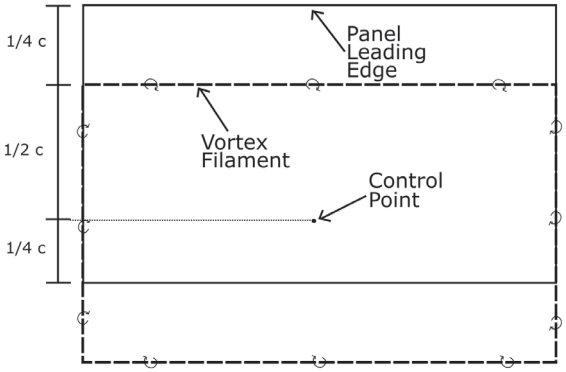

FIGURE 1 Diagram of a vortex ring overlaid on top of an aerodynamic panel. C, The chord of the panel. The solid line represents the aerodynamic panel, whereas the dashed line represents the vortex ring. The dot represents the control point, and the arrows represent the direction of flow around the vortex filaments

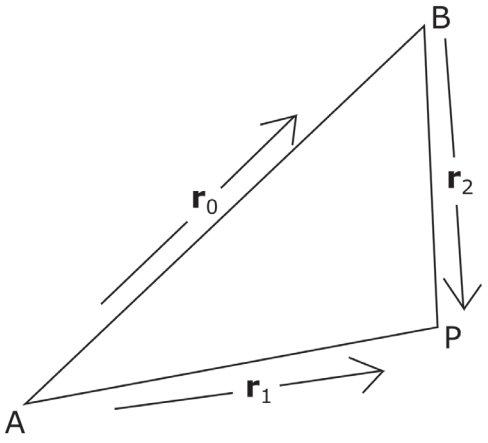

FIGURE 2 Diagram of the vectors required to calculate the effect of a vortex filament spanning from "A" to "B" on the velocity at a point "P"

Each filament on the ring is then placed anti-clockwise around the center of the ring such that each filament induces velocity in the same direction inside of the ring (see Figure 1). Since each filament for the same ring has the same strength, this also means that a rectangular ring will induce velocity in the opposite direction outside of the ring.

As shown in Figure 3, the wing is discretized into a structured array of panels; in this case only the vortex rings are shown for clarity. At each time step, a row of wake panels are shed at the trailing edge of the wing according to the equation:

$$\boldsymbol{X}_w = \boldsymbol{X}_r + \boldsymbol{V} \cdot \Delta t, \tag{2}$$

where $\boldsymbol{X}_r$ is the trailing edge vortex ring corner point; $\boldsymbol{X}_w$ is the corresponding wake vortex ring corner point; $\boldsymbol{V}$ is the air velocity at the point $\boldsymbol{X}_r$ relative to the wing (including the kinematics of the wing itself); and Δt is the time step. This means that the point $\boldsymbol{X}_w$ is essentially at the point where an air particle starting at $\boldsymbol{X}_r$ would end up after one time step relative to the wing. $\boldsymbol{X}_y$ in Figure 3 is found using the same time integration scheme from $\boldsymbol{X}_w$.

This process is repeated for every time step in the analysis, meaning that both the number of calculations required to determine the velocity at a point increases; and the number of points in the domain where the velocity must be known also increases. This gives rise to both the order of computations $O(n^2)$ and a large n in the analysis, which is what leads to the method's large overall computational cost.

Given that the required number of time steps in an analysis will generally be much larger than the number of chordwise panels, most of the computational cost of the analysis is due to the wake calculations, which is the part of the analysis which the double-tree method and the Willis et al[14] method mainly address.

A more detailed description of the UVLM, including discretization of the wing and wake generation, can be found in Reference 1. The next section will discuss the construction of the filament tree, which uses agglomerated rings based on UVLM theory to solve the problem in $O(n \cdot \log n)$ time.

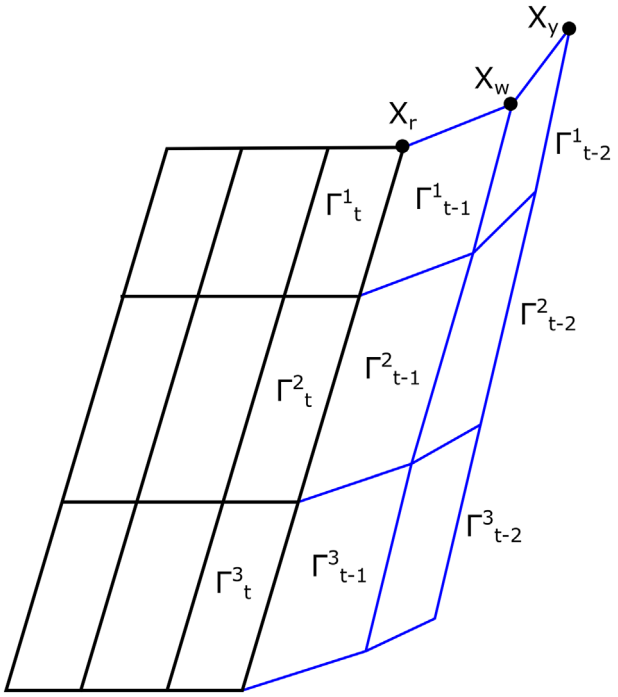

FIGURE 3 Diagram of wake shedding in the UVLM. Black lines represent the wing, whereas blue lines represent the wake. $\Gamma^j{}_t$ is the strength of ring j at time t [Colour figure can be viewed at wileyonlinelibrary.com]

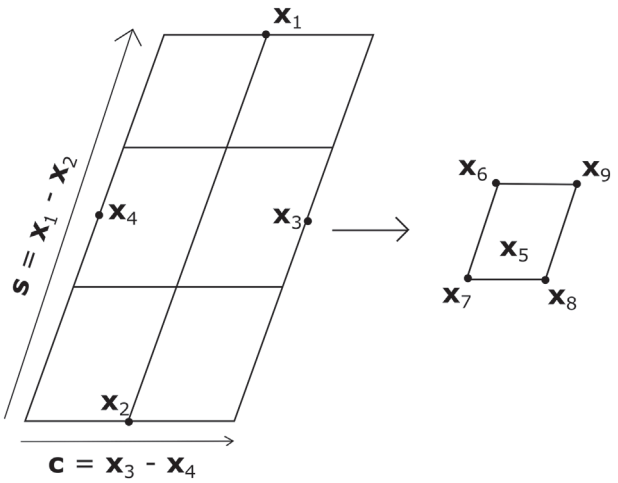

FIGURE 4 Agglomeration of vortex rings. $\boldsymbol{X}_1$ through $\boldsymbol{X}_4$ are the mean coordinates of the points on their respective edges. The dimensions of the agglomerated ring are then found by dividing the vectors $\boldsymbol{C}$ and $\boldsymbol{S}$ by the number of rings in the respective directions. The center of the agglomerated ring $\boldsymbol{X}_5$ is the mean of points $\boldsymbol{X}_1$ through $\boldsymbol{X}_4$

3 | FILAMENT TREE

In order to reduce the order of computations, far-field vortex rings are agglomerated into one ring, constructed such that the difference in the velocity field reduces as the distance to the agglomerated ring increases. This can be done by constructing a ring whose size is the average of the agglomerated rings and whose strength is the sum of the agglomerated rings.

The method of agglomeration in the double-tree method is based upon the assumption that the rings are arranged into a regular two-dimensional structure. Points on the outside edges of an array of vortex rings are used to find the average dimensions of the rings, as illustrated in Figure 4.

The span of the agglomerated ring is defined as the mean span of the rings it agglomerates, and can be calculated by taking the mean edge positions of the rings; finding the difference between them; and dividing the result by the number of spanwise rings. The mean top edge position is calculated from

$$\boldsymbol{X}_1 = \frac{\sum \boldsymbol{X}_{\text{top}}}{N_{\boldsymbol{X}_{\text{top}}}}, \tag{3}$$

where $\boldsymbol{X}_{\text{top}}$ is the set of points along top edge and $N_{\boldsymbol{X}_{\text{top}}}$ are the number of points along the top edge. The same process can be followed to find points $\boldsymbol{X}_2$ through $\boldsymbol{X}_4$ (Figure 4). The span vector of the agglomerated ring is then calculated using

$$\boldsymbol{S} = \frac{\boldsymbol{X}_1 - \boldsymbol{X}_2}{N_{\text{span rings}}}, \tag{4}$$

where $N_{\text{span rings}}$ is the number of vortex rings in the spanwise direction. The same process can be performed to calculate the chord vector, $\boldsymbol{C}$. The center point of the agglomerated ring is

$$\boldsymbol{X}_5 = \frac{\sum_{i=1}^{4} \boldsymbol{X}_i}{4}. \tag{5}$$

The points $\boldsymbol{X}_6$ through $\boldsymbol{X}_9$ are used to define the agglomerated ring. They are found from the center point $\boldsymbol{X}_5$ as follows:

$$\boldsymbol{X}_{6\ldots9} = \boldsymbol{X}_5 \pm \frac{\boldsymbol{S}}{2} \pm \frac{\boldsymbol{C}}{2}. \tag{6}$$

Finally, the strength of the agglomerated ring is

$$G_{\text{agg}} = \sum_{i=1}^{n} G_i, \tag{7}$$

where i is the index of a vortex ring, n the number of rings, and Γ is the vortex strength defined in Equation (1). To verify the agglomeration method, the difference in the velocity field compared with the UVLM is plotted for four vortex rings agglomerated into one.

Given that the difference is shown to drop off rapidly with distance (Figure 5), this agglomeration method can be considered a valid representation of far-field vortex rings. When the number of agglomerated rings is squared, the width of the difference field profile increases by a factor slightly above 2, otherwise retaining the same shape. This suggests a strongly linked relationship between the difference field and the number of agglomerated rings.

In order for a tree structure to be built, there must also be a method of agglomerating a group of up to four agglomerated rings. First, the total strength of the agglomerated rings is summed according to (7). Then, both the center point and the dimensions of the new agglomerated ring are found by the weighted average of the corresponding child ring data,

$$X_c = \frac{\sum_{i=1}^{n} X_{c,i} \cdot G_i}{G_{\text{agg}}}, \tag{8}$$

$$S_{\text{agg}} = \frac{\sum_{i=1}^{n} S_{c,i} \cdot G_i}{G_{\text{agg}}}, \tag{9}$$

where i is the index of an agglomerated child vortex ring, n the number of child rings, and Γ_{agg} is the agglomerated vortex strength defined in Equation (7). X_c is the center point of an agglomerated ring ($\boldsymbol{X}_5$ in Figure 4) and S_{agg} its span. A similar method is used for the agglomerated chord length.

Using this agglomeration method, a tree structure can be generated that allows for different levels of agglomeration around any given point such that more vortex rings are agglomerated as the distance from the query point (a point in the domain whose flow velocity must be calculated) increases.

A recursive algorithm is used to construct the tree (Figure 6), which passes sub-sectional information down to the leaves, and agglomeration information up to the root. To construct the tree, the root cell (number 1 in Figure 6) is given two lists containing the spanwise and chordwise identities of the panels, as well as the locations of all points that define the rings. The first step is to find the minimum and maximum values of the vortex ring locations contained within the cell in each orthogonal direction, which is later used to calculate whether the cell is near or far-field.

Next, the total number of rings N_{rings} (Equation (10)) is checked to see if it exceeds the *bucket size* (Willis et al. recommend between 10 and 15).[14] The bucket size determines the maximum size of a leaf cell, in this case the maximum number of vortex rings in the cell. If the size is too small, the tree will be large and searching it will be inefficient; whereas if the bucket size is too large then the tree will not have enough levels of refinement for the algorithm to exploit.

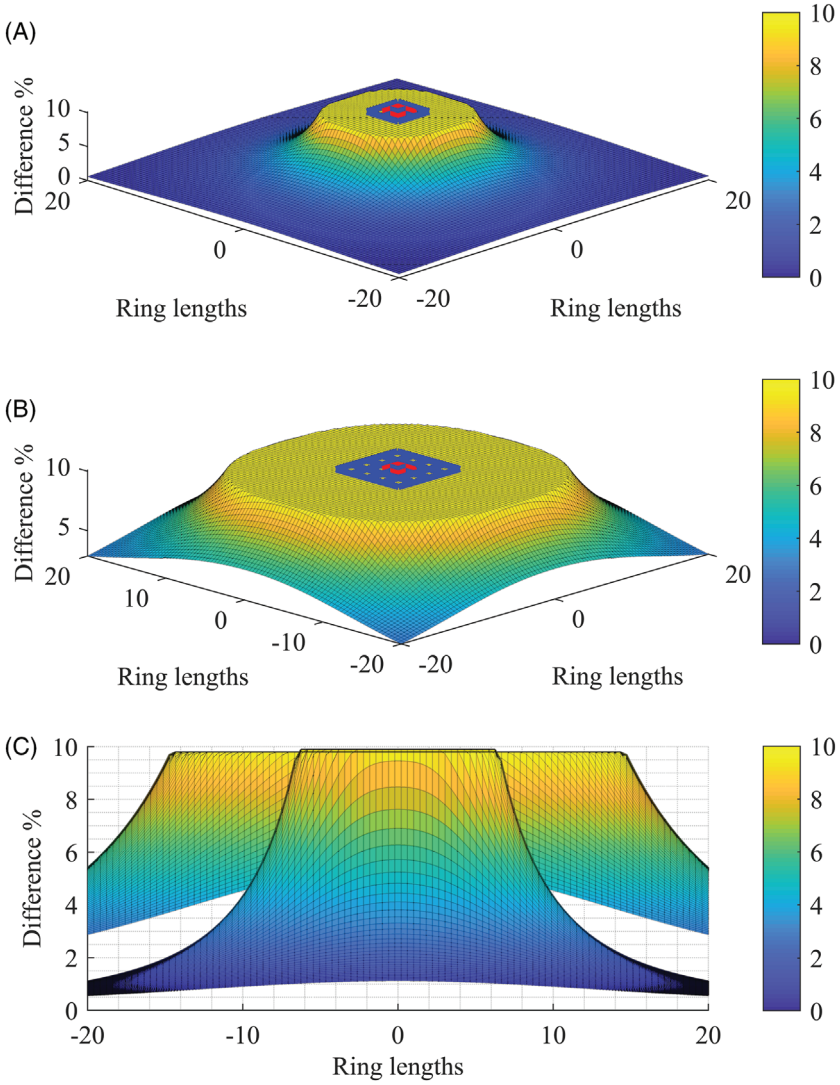

FIGURE 5 A, Plot of the difference in the velocity field around an agglomerated vortex ring (red). The high yellow area has a difference of 10% or greater, which extends out to seven ring lengths from the center of the original four rings (blue). The difference can be seen to drop off rapidly as the distance to the rings increases. B, The same plot of the difference field, but with a 16-ring agglomeration instead of 4. The area with a 10% difference or greater extends to about 15 ring lengths from the center. C, Both A and B are plotted together, with a slice through the middle and viewed from the side

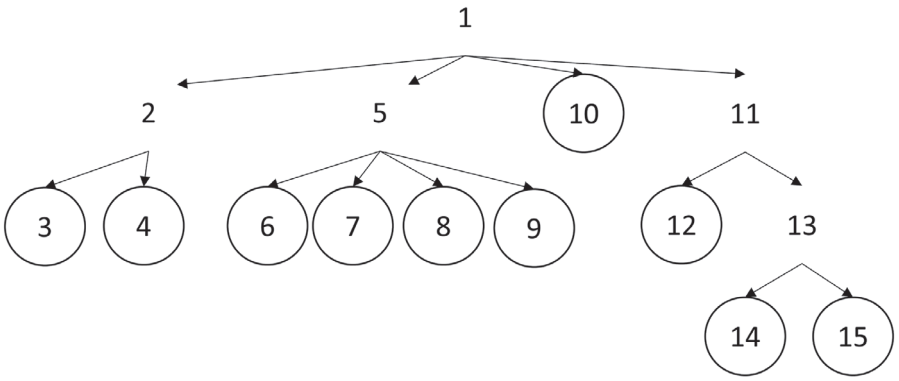

FIGURE 6 Example tree structure. The cell (1) contains all vortex rings, and splits them into four child cells by proximity. Leaf cells (circled) apply the direct agglomeration method to the rings contained therein, whereas all other cells use the agglomerated-ring method

Generally speaking, though, the performance is not sensitive to small differences in bucket size, so any value in the aforementioned range should be suitable.

$$N_{\text{rings}} = N_{\text{chordwise}} \cdot N_{\text{spanwise}} > N_{\text{bucket size}}, \tag{10}$$

where $N_{\text{chordwise}}$ and N_{spanwise} are the number of chordwise and spanwise vortex rings, respectively. If (10) is false, the cell is a leaf cell. The leaf cell stores the identities of the rings it contains, as well as the agglomeration of the rings according

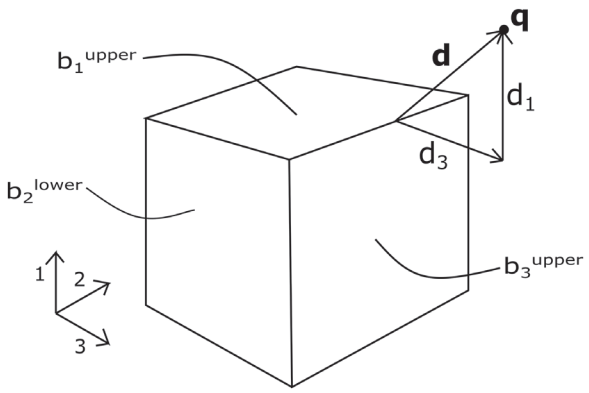

FIGURE 7 Diagram showing how **d** is calculated in order to determine the cutoff distance criterion. In this example, **q** is in between b_2^{lower} and b_2^{upper}, but greater than b_1^{upper} and b_3^{upper}

to Equations (3) to (7). The identity of the rings within the leaf cell are calculated using the chordwise and spanwise lists, along with information regarding the original size of the list.

If Equation (10) is true, the algorithm must recurse. It was found empirically that the algorithm performs more efficiently if the aspect ratio of each cell is close to 1; therefore, the recursion of the algorithm changes according to the following criterion:

$$\text{Case 1} : N_{\text{chordwise}} > 2 \cdot N_{\text{spanwise}}, \tag{11}$$

$$\text{Case 2} : N_{\text{spanwise}} > 2 \cdot N_{\text{chordwise}}. \tag{12}$$

If case 1 is true, two child cells are created, each containing half of the list of chordwise panels. Similarly, if case 2 is true, the child cells will contain half of the list of spanwise panels each. If neither is true, four child cells are created by dividing the panels both chordwise and spanwise. After recursing, the cell agglomerates its children according to Equations (7) to (9).

To calculate the velocity induced by the filament tree on a point, another recursive algorithm is used. Starting with the root cell, the *cutoff distance* criterion is calculated with the distance from the query point to the boundary of the cell.

$$\|\boldsymbol{d}\| < u, \tag{13}$$

$$d_i = \begin{cases} q_i - b_i^{\text{lower}}, \text{if } q_i < b_i^{\text{lower}} \\ q_i - b_i^{\text{upper}}, \text{if } q_i > b_i^{\text{upper}} \\ 0, \text{otherwise} \end{cases}, \tag{14}$$

$$b_i^{\text{lower}} = \min(\boldsymbol{x}_i); b_i^{\text{upper}} = \max(\boldsymbol{x}_i), \tag{15}$$

where $\boldsymbol{q}_i$ is the ith coordinate ($i = 1, 2, 3$) of the query point; $\boldsymbol{x}_i$ is a vector containing the ith coordinate of all vortex ring corner points in the cell; and u is the cutoff distance. Essentially, $\boldsymbol{d}$ is the shortest distance from the query point to the surface of a cuboid drawn around all of the vortex rings in the cell, except inside of the cuboid itself where $\boldsymbol{d}$ is zero. This is illustrated in Figure 7.

If Equation (13) is true and the cell is not a leaf, each of the cell's children are searched in the same way. If the cell is a leaf, then the rings contained within the cell are calculated according to the basic UVLM theory without agglomeration. If Equation (13) is false, the cell's agglomerated ring is used to calculate the velocity.

4 | QUERY POINT TREE

The query point tree agglomerates together multiple points in the domain whose velocity must be calculated, referred to in this paper as *query points*. If a vortex filament is sufficiently far away from a group of query points, its influence will be

calculated once for the average position of the query points, and the result added to the total velocity of each point in the group.

The query point tree is constructed in exactly the same way as the filament tree, except that the only agglomeration information required for a leaf cell is the average position of the query points contained therein; all other cells take the average position of their child cells,

$$\boldsymbol{x}_C{}^{\text{leaf}} = \frac{\sum \boldsymbol{x}_q}{N_{\boldsymbol{x}_q}}, \tag{16}$$

$$\boldsymbol{x}_C{}^{\text{branch}} = \frac{\sum \boldsymbol{x}_C{}^{\text{child}}}{N_{\boldsymbol{x}_C{}^{\text{child}}}}, \tag{17}$$

where $\sum \boldsymbol{x}_q$ is the sum of the query point positions in the leaf cell; $N_{\boldsymbol{x}_q}$ is their number; and $\boldsymbol{x}_C$ is their average position. $\boldsymbol{x}_C{}^{\text{leaf}}$ is then the position of the leaf cell. Similarly, $\sum \boldsymbol{x}_C{}^{\text{child}}$ is the sum of the positions and $N_{\boldsymbol{x}_C{}^{\text{child}}}$ the number of the children of a cell, which can also be leaves, and $\boldsymbol{x}_C{}^{\text{branch}}$ is the position of a branch cell.

This average position $\boldsymbol{x}_C$ is used as the query point in the UVLM velocity calculations. Similar to the filament tree, a distance cutoff criterion is used to determine whether a cell should be searched when calculating the velocities at the query points, the only difference being that the distance is measured at both ends of the filament in question and the minimum value used.

During velocity calculation, two separate variables are required to be used to store the velocity at the query points in order for the operation to be efficient. The first variable stores velocity values for all non-agglomerated operations, and the second stores the velocity induced for each cell in the tree.

$$\boldsymbol{V}_{\text{non-agg}} = \left[\!\!\left[\right]\!\!\right]_{3\times n}, n = \text{number of query points}, \tag{18}$$

$$\boldsymbol{V}_{\text{agg}} = \left[\!\!\left[\right]\!\!\right]_{3\times c}, c = \text{number of cells in query tree}, \tag{19}$$

where $\left[\!\!\left[\right]\!\!\right]_{i\times j}$ denotes a matrix of size $i \times j$.

After the analysis, the tree cell velocity can be utilized by accessing the chordwise and spanwise lists for each cell in order to identify which query points the velocity should be summed over. This is measured to be a negligible part of the total processing time.

5 | DOUBLE TREE

5.1 | Implementation

The double-tree method utilizes both the filament and query point trees in tandem in order to calculate the velocity at the query points. Both trees are constructed in exactly the same way; the only difference in the double-tree method is how the velocity is calculated. Starting at the root of both trees, the cutoff distance criterion is calculated using

$$d_i = \begin{cases} b_i{}^{\text{lower-filament}} - b_i{}^{\text{upper-query}}, \text{if } b_i{}^{\text{lower-filament}} > b_i{}^{\text{upper-query}} \\ b_i{}^{\text{lower-query}} - b_i{}^{\text{upper-filament}}, \text{if } b_i{}^{\text{lower-query}} > b_i{}^{\text{upper-filament}} \\ 0, \text{otherwise} \end{cases}, \tag{20}$$

where b_i is defined in the same way as in each tree individually (Equation (15)).

The following logic is used to determine how the two trees should be searched.

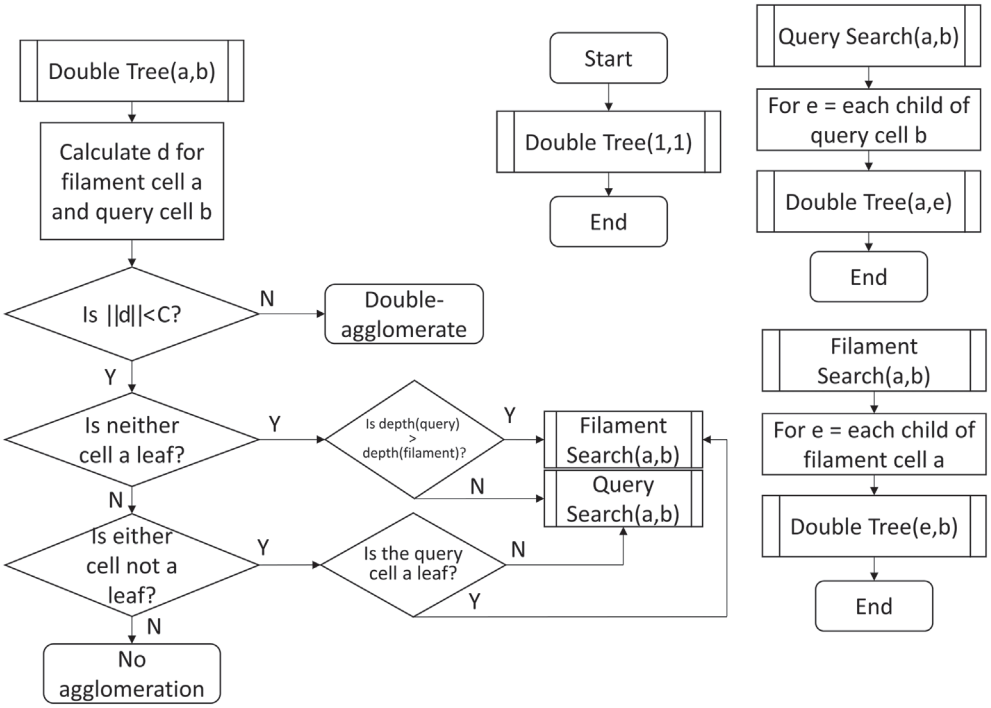

FIGURE 8 Flowchart detailing the recursive tree search operation of the double-tree method

As can be seen in Figure 8, at the start of the algorithm, the *Double Tree* subprogram is called, with parameters [1,1]. These parameters indicate that cell 1, which is the root, of each tree should be compared. $\boldsymbol{d}$ from Equation (20) is calculated to determine if the tree cells are far enough apart to perform a double agglomeration, where the influence of an agglomerated vortex ring (Equations (6)-(9)) is calculated on an agglomerated query point (Equations (16)-(19)). Thus, the computational advantage of both methods is exploited simultaneously. If the cells are closer than the cutoff distance, several checks are performed to determine how the tree should be searched. If neither cell is a leaf cell, the tree with the least depth, that is, the closest to the root is searched. This is done by calling another subprogram, either *Query Search* or *Filament Search*, which in turn call *Double Tree* for each child cell of the tree cell to be searched. Otherwise, if one of the cells is a leaf and the other is not, the non-leaf cell is searched, and finally, if both cells are leaves, the basic UVLM is performed. Using this procedure, all query-filament interactions in the domain are calculated with either double agglomeration or non-agglomeration.

5.2 | Order of computations

The double-tree algorithm is shown to be $O(n)$ for a simple case in the following way. Suppose there is a regular grid of $n = 4^x$ vortex rings, each containing one query point in its center. For a cutoff distance of 0, the root of each tree requires further refinement, since they overlap. The children of each cell create a 2×2 grid of overlapping filament tree and query tree cells requiring three computations per cell, resulting in 24 total computations, as shown in Figure 9.

Following the query-filament cell pair (A) in Figure 9, the interactions with cell pairs (B) to (D) have been calculated with double agglomeration, meaning the only remaining interaction is (A)'s filament tree cell with its query tree cell. Since the two cells in (A) overlap, further refinement is required, but because the grid is regular and the trees are constructed indexically, (A) will have exactly four child cells, whose arrangement is equivalent to that given in Figure 9. When a leaf is reached in this recursive manner, it will contain four vortex rings and control points. Thus, for every cell in the trees, 24 computations are required to find the solution. Since the total number of cells in this case is $\sum_{i=0}^{x-1} 4^i$, where $x = \log_4 n$, the total computations in this case is $24 \sum_{i=0}^{(\log_4 n)-1} 4^i$, which can be simplified to $8n - 8$. Thus, the order of computations is $O(n)$ for a cutoff distance of zero. Intuitively, as the cutoff distance is increased to infinity, the order of computations increases to the $O(n^2)$ of the basic UVLM. However, based upon the example in this section and empirical evidence shown in Section 6.1, the authors conjecture the double-tree method is $O(n)$ for all interactions beyond the cutoff distance.

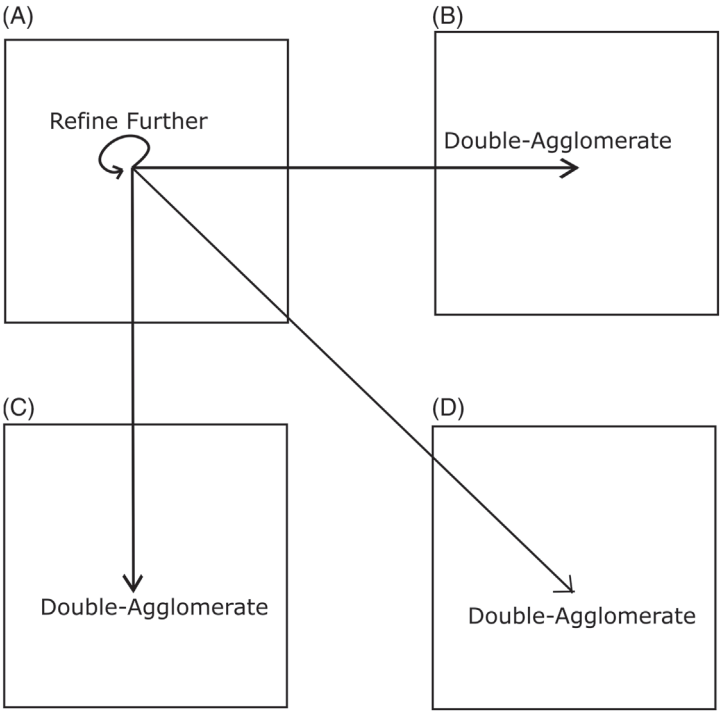

FIGURE 9 Diagram illustrating how a pair of filament and query tree cells (A) interacts with other query-filament tree cell pairs (B)-(D) in the domain. One double-agglomeration procedure is performed for each other cell pair. The query tree cell and the filament tree cell overlap in (A), so they must be refined further

6 | RESULTS

All test cases are implemented in MATLAB and run on the same machine, which has a 2.6GHz i7 processor. All test-cases are run non-parallel in order to fairly measure the order of computations and the computation time of each method, due to the differing parallelization schemes between the double-tree method and the other methods. The *parallelization* test case is the exception, which is run in MATLAB on a high-performance computing cluster with a specified parallel number of cores.

6.1 | Order of magnitude

The first test is performed to measure the scaling of each tree method for problems with increasing order of magnitude. An $m \times m$ regular grid of vortex rings is generated, and the velocity at the corners of every ring is then calculated. The wall clock time for the filament tree, query point tree, and double-tree methods to create the trees and calculate the velocity field are compared for a cutoff distance of 0 and a number of vortex rings $N \approx 10^x$, $x = [1, 2, 3, 4, 5]$. The iteration time is averaged across three runs, each of which exhibited the same trend. These results are not compared with the base UVLM due to its excessive computation time for large N.

As shown by Figure 10, the double-tree method's iteration time per N decreases as N increases. This is likely due to constant overhead (such as tree generation) contributing a lower fraction of the total time as N increases, while the velocity calculation time per N remains constant. For $N \approx 10$ vortex rings, the overhead dominates the computational cost, leading to an exceptionally high computational cost per vortex ring; however, the total iteration time for an analysis of this size is negligible anyway. The other two individual tree methods can be seen to have an increasing iteration time per N despite the same diminishing overhead effect, due to the log$n$ part of the velocity calculation scaling. Based upon this evidence, it is reasonable to conclude that the double-tree method is $O(n)$ and that the individual-tree methods are $O(n \cdot \log n)$. It can also be concluded that the double-tree method only provides a superior computational cost compared with the other tree methods for analyses with approximately 1000 or more vortex rings.

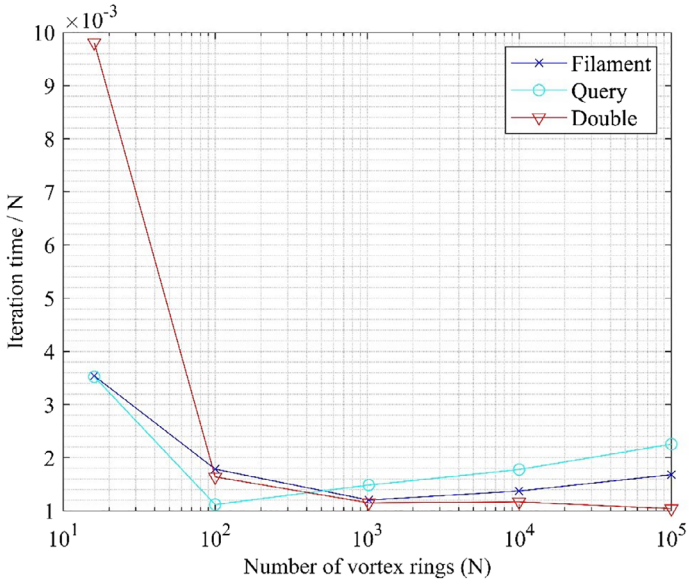

FIGURE 10 Plot of the time for an iteration to complete per vortex ring against the number of vortex rings [Colour figure can be viewed at wileyonlinelibrary.com]

6.2 | Accuracy vs speed

The next test measures the effectiveness of the tree methods by measuring computation time and accuracy. The original UVLM is compared with the filament tree method; the query point tree method; and the double-tree method. The UVLM is verified against the Katz and Plotkin[1] data for a flat plate undergoing sudden acceleration. The accuracy is measured as the difference in the distribution of the magnitude of the forces between the UVLM and the tree methods, using the formula:

$$\Delta_F = \frac{\|\boldsymbol{F}^{\text{tree}} - \boldsymbol{F}^{\text{UVLM}}\|_2}{\|\boldsymbol{F}^{\text{UVLM}}\|_2}, \tag{21}$$

where $\boldsymbol{F}$ is a $3 \times N$ matrix of force values calculated on the wing panels.[1]

The setup is of a flat plate undergoing sudden acceleration. In order for the wake to grow sufficiently in size with each time step, the plate has 5 chordwise and 200 spanwise panels, and has an aspect ratio of 40. The tree bucket size is 15 in every case; the time step duration is 0.003 second; the air density is 1 kg/m^3; the UVLM shedding ratio1 is 0.25; and the ground speed is 20 m/s. Results are compared for the same cutoff distances; however, each tree method can have differing levels of agglomeration for the same cutoff distance. The computation time is measured as the "wall clock" time that passes while each time step completes, and is averaged across three separate runs of each method and test case.

As shown in Figure 11, for the same cutoff distance for this test case, the filament tree method has the lowest computational cost, followed by the double-tree method, and then the query tree method. For a cutoff distance of one ring length, a speedup of over one order of magnitude is achieved, with a force distribution difference of approximately 20% for the query tree and double-tree methods. This gives an idea of the maximum achievable speedup for a problem of this size. The speedup drops to one order of magnitude for a cutoff distance of two rings, but with the difference dropping to 10% for the query and double tree methods.

It can be seen that a cutoff of one ring for the query and double-tree methods provides both less difference and less computational cost than the filament tree with a two ring cutoff. This same trend continues as the cutoff distances are doubled. For a cutoff distance of eight rings, the filament tree overtakes the query point tree in terms of accuracy, but the double-tree method remains consistently high in accuracy across all runs. The difference appears to drop off faster than the computational efficiency as the cutoff distance is increased.

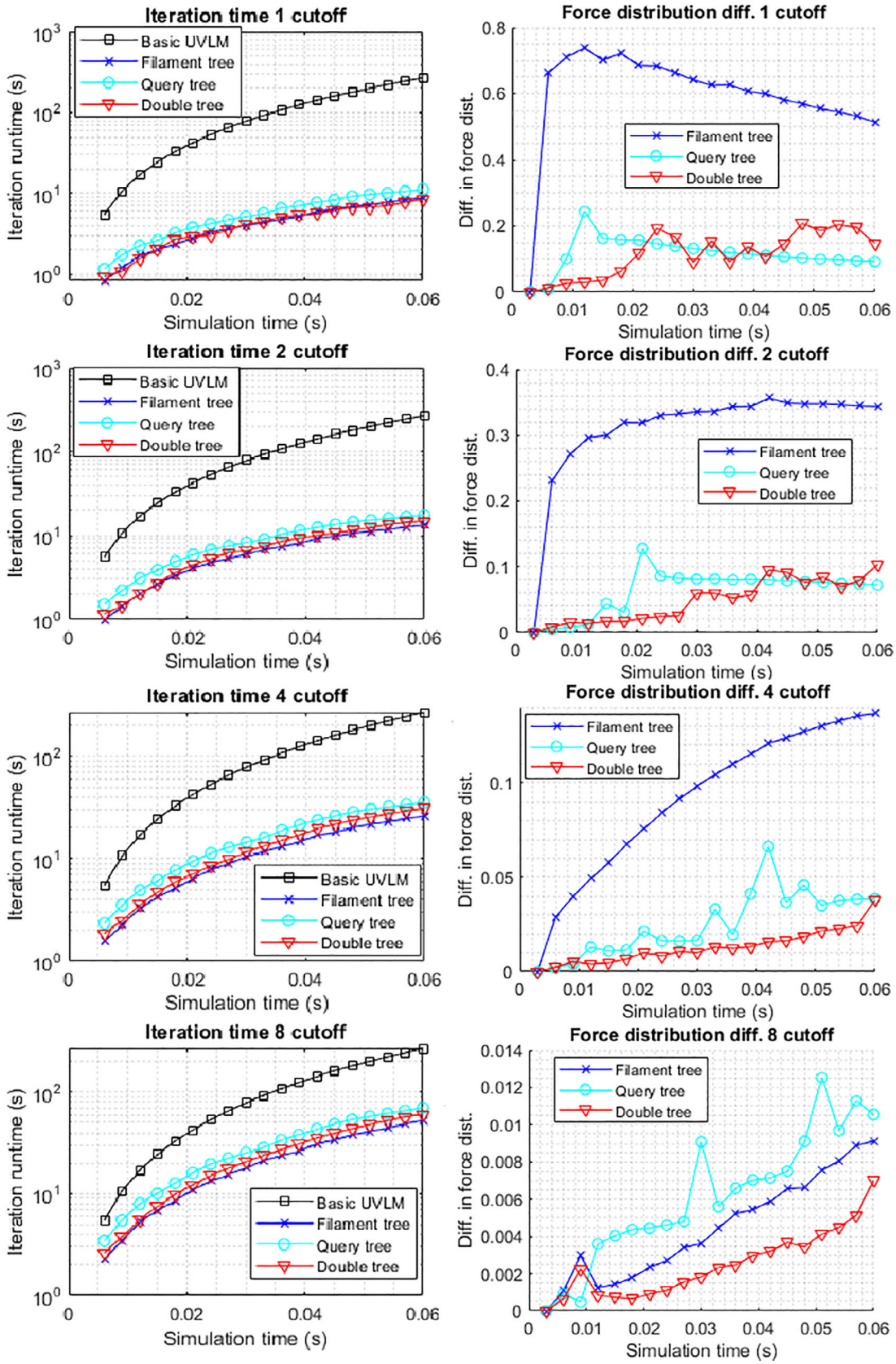

FIGURE 11 Combined plots of force distribution difference and iteration runtime for different cutoff distances. The cutoff distance is measured in vortex ring lengths [Colour figure can be viewed at wileyonlinelibrary.com]

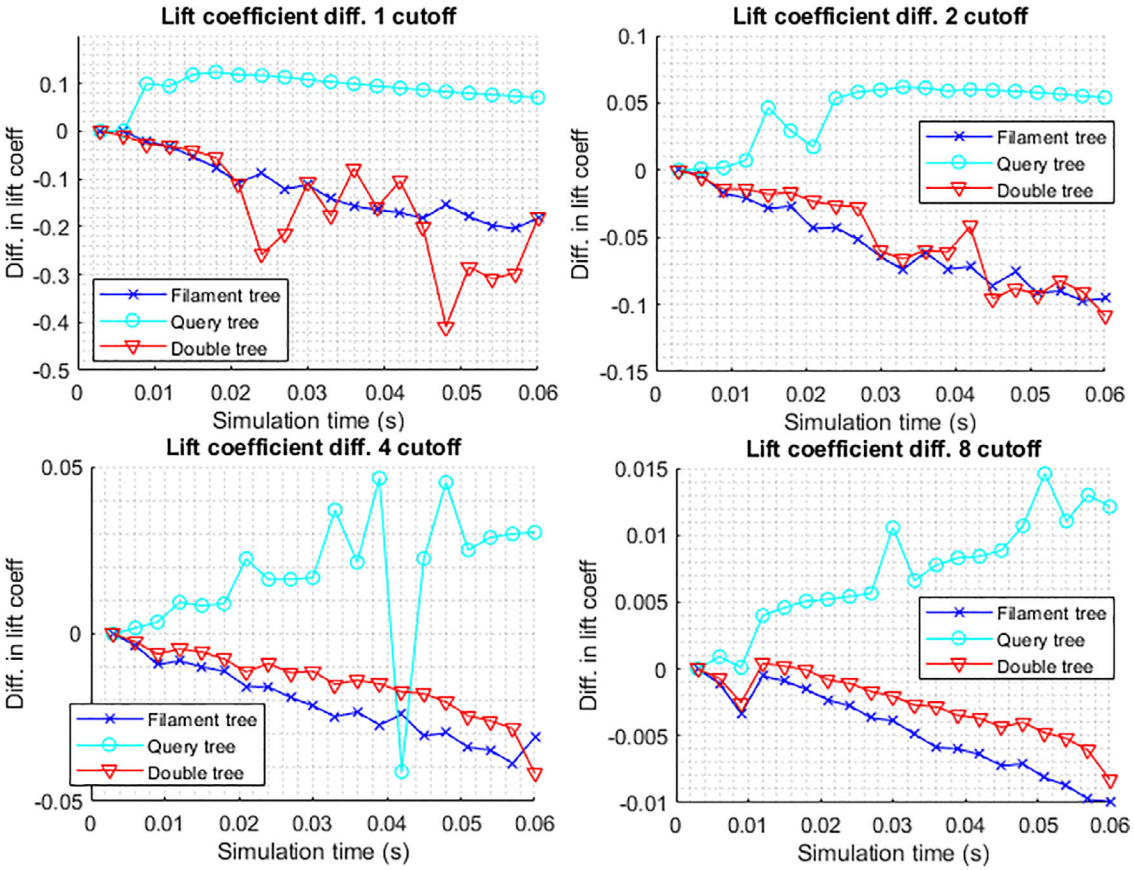

FIGURE 12 Difference in lift coefficient on the wing at each time step for different cutoff distances [Colour figure can be viewed at wileyonlinelibrary.com]

Based upon these tests, it may be concluded that all tree methods perform similarly in terms of computational efficiency; however, the double-tree method has a consistently lower difference for the same cutoff distances compared with the other methods.

6.3 | Lift coefficient difference

The difference in the total lift coefficient on the wing for each tree method is evaluated against the basic UVLM using the formula:

$$\Delta_{C_L} = \frac{{C_L}^{\text{tree}} - {C_L}^{\text{UVLM}}}{{C_L}^{\text{UVLM}}}, \tag{22}$$

where C_L is the total lift coefficient on the wing. The test case used is the same as the *accuracy vs speed* test.

As shown in Figure 12, the query point tree tends to overestimate the lift coefficient, whereas the other two trees tend to underestimate it. The filament tree has the most consistent difference profile, achieving 20%, 10%, 5%, and 1% difference at the respective increasing cutoff distances. At lower cutoff distances, the double-tree method experiences a large variation in the difference; however, it generally outperforms the filament tree at larger cutoff distances. At small cutoff distances, the query point tree has the lowest difference, however its difference appears to vary more and the method performs worse at larger cutoff distances than the other tree methods. Large difference variation could be caused by a combination of vortex stretching effects, singularity effects, and different rounding of elements into tree cells between iterations. For a cutoff distance of zero (where agglomeration is always used between non-overlapping cells), the absolute difference can reach well over 100% due to these effects.

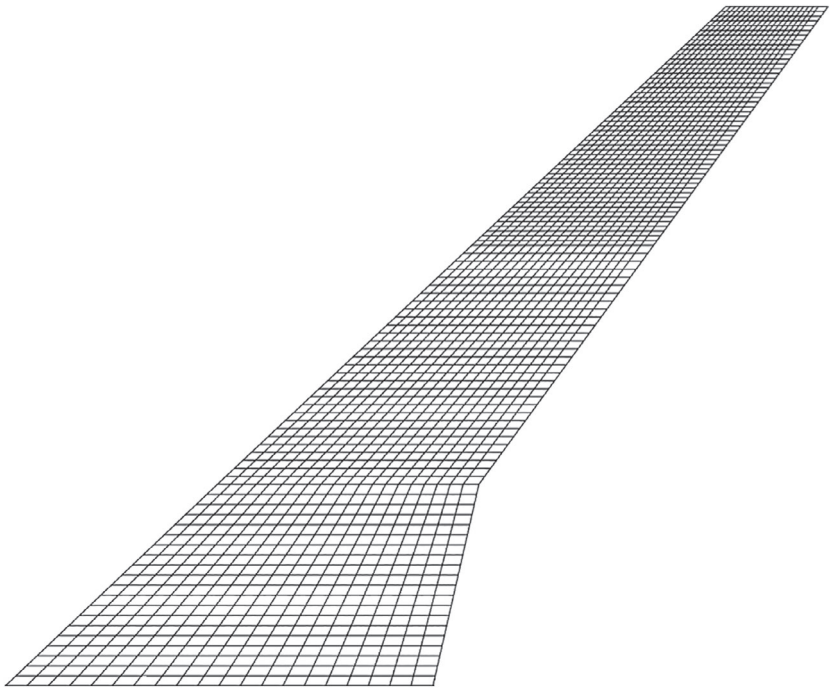

FIGURE 13 Plot of the geometry used for the swept tapered wing analysis. The planform is derived from the NASA common research model wing[20]

6.4 | Swept tapered wing

In order to verify the efficacy of the tree methods for more complex geometries, an analysis is performed on a swept tapered flat plate whose geometry is shown in Figure 13.

The plate contains 20 chordwise and 200 spanwise panels. The non-dimensional number $\frac{c}{U_\infty \Delta t}$ is kept consistent with the other test cases (where c is the root chord) by increasing the time step to 0.03357 second. The aspect ratio can be calculated with $\text{AR} = \frac{s^2}{A}$, where s is the wingspan and A is the area of the wing. The span is measured as the distance from root to tip in the direction perpendicular to the root, in this case 26.14 m, which is doubled to 52.28 m for the total wingspan. The area of the wing is measured as the sum of the area of the panels, at 190.37 m^2, making the aspect ratio of the wing 14.36. All other parameters are the same as in the *accuracy vs speed* test case (Section 6.2).

Comparing Figure 12 with Figure 14, all of the general trends are repeated, except for less variance in the difference and an increased magnitude of difference in this case. These trends are also observed in the force distribution difference (not shown). The computation time saved by the four and eight cutoff distance analyses are approximately 79% and 43%, respectively. The cutoff distance is measured in ring lengths; however, the rings at the root were chosen for this measurement, which are approximately four times longer than the rings at the tip.

Overall, the general trends in the difference are similar, although the magnitude of the difference and the computation time are worse in this case than in the other test cases. However, the time-saving for this example is still significant when comparing the tree methods to the basic UVLM.

6.5 | Analysis size effects

The effect of different wing sizes is investigated. A flat plate with an aspect ratio (AR) of 40 (5 chordwise and 200 spanwise square panels) undergoing sudden acceleration is investigated for 20 time steps. It is compared against plates of AR 20, 10, and 5. The other AR plates are created by truncating the AR 40 plate, such that the number of spanwise panels is reduced. The differences are calculated in the same way as stated previously (Equations (21) and (22)), taking the maximum absolute value across the 20 time steps. The raw data values can be found in Appendix A.

As can be seen in Figure 15, the double-tree method has the smallest maximum force distribution difference compared with the other tree methods in every case. All tree methods show that the force distribution difference is reduced as the cutoff distance increases; however, the filament tree method has more sensitivity to the size of the problem than the other tree methods.

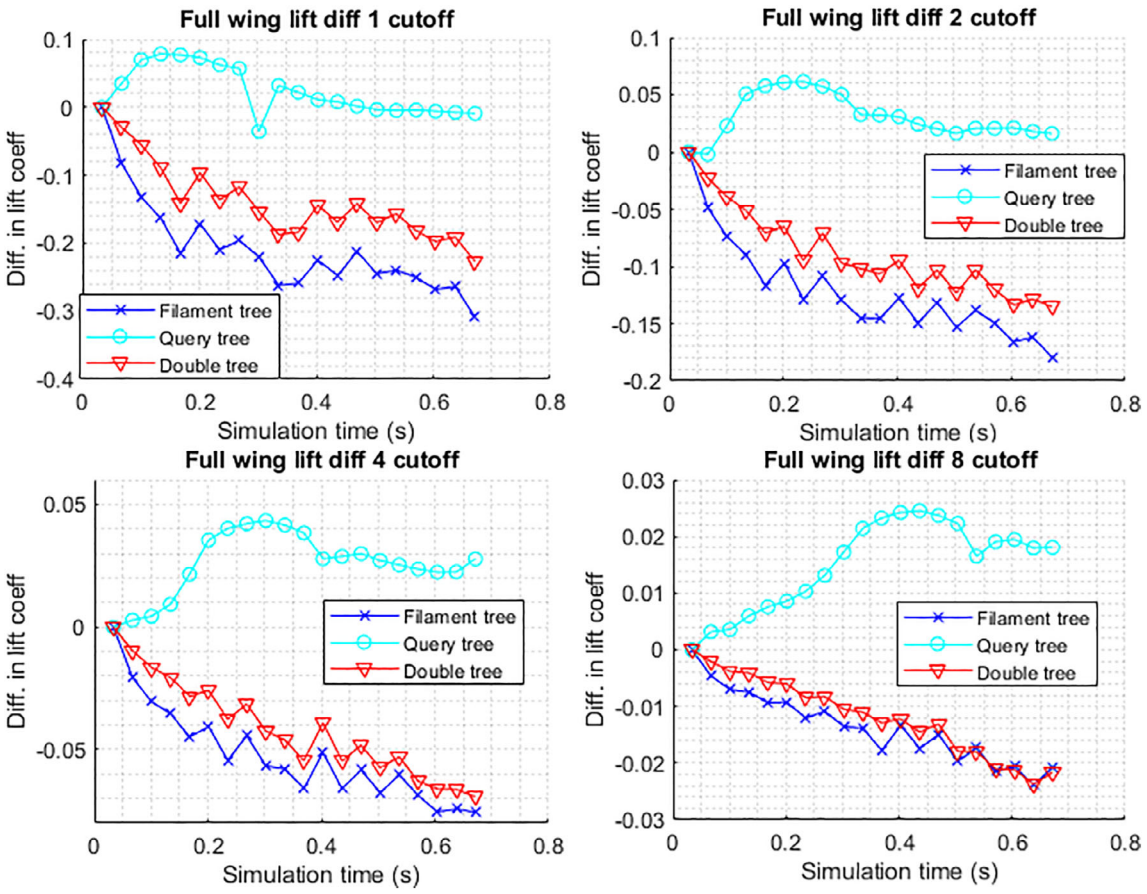

FIGURE 14 Difference in lift coefficient on the wing at each time step for different cutoff distances [Colour figure can be viewed at wileyonlinelibrary.com]

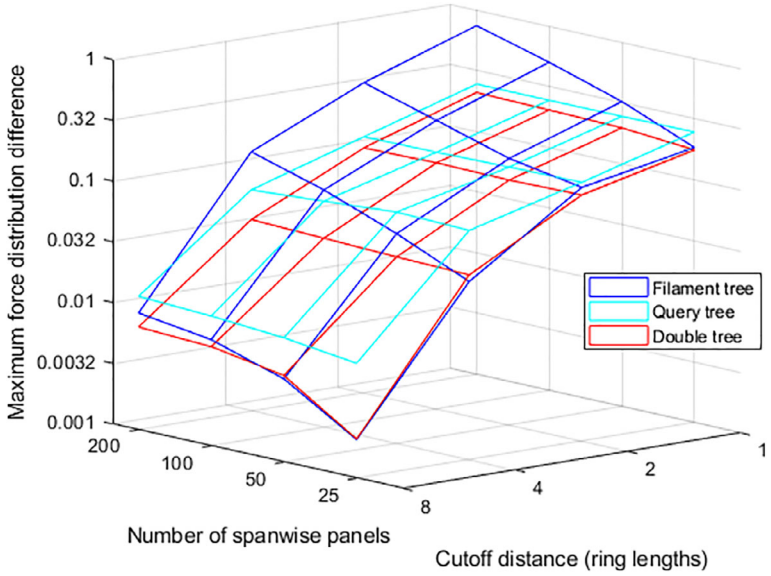

FIGURE 15 Log plot of the variation of the maximum force distribution difference across 20 time steps of the three tree methods compared with the basic UVLM, with the number of spanwise panels and the cutoff distance [Colour figure can be viewed at wileyonlinelibrary.com]

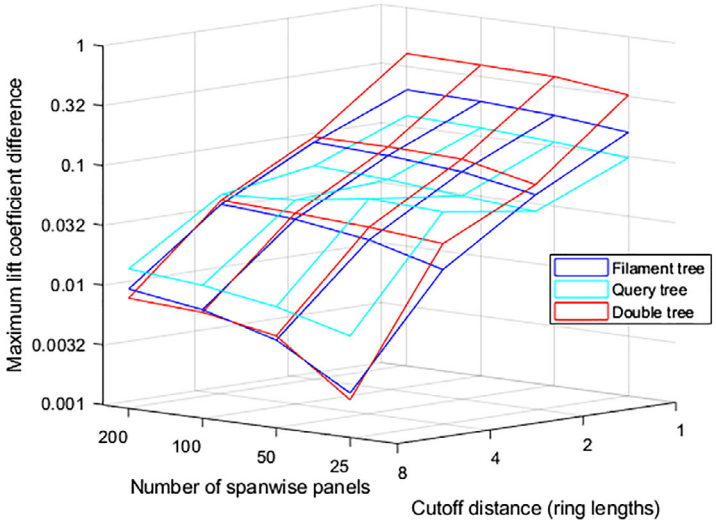

FIGURE 16 Log plot of the variation of the maximum magnitude lift coefficient difference across 20 time steps of the three tree methods compared with the basic UVLM, with the number of spanwise panels and the cutoff distance [Colour figure can be viewed at wileyonlinelibrary.com]

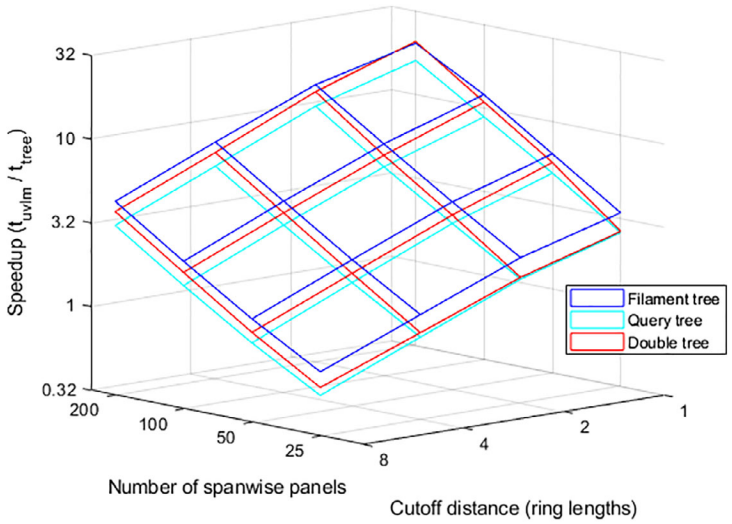

FIGURE 17 Log plot of the variation of the speedup of the three tree methods compared with the basic UVLM, with the number of spanwise panels and the cutoff distance. The speedup is measured as the total computational time of the tree method divided by the total computational time of the UVLM [Colour figure can be viewed at wileyonlinelibrary.com]

Figure 16 shows that the double tree has larger lift coefficient difference for small cutoff distances compared to the other tree methods; however for large cutoff distances it moves towards having the smallest difference. As in Figure 15, the maximum difference decreases as cutoff distance increases and the number of panels decreases. This is because a larger portion of the domain is non-agglomerated for larger cutoff distances in smaller problems, bringing the overall methods closer to the UVLM.

As can be seen in Figure 17, the computational speedup is much more strongly correlated with the size of the problem than the differences in force distribution or lift coefficient. This suggests that for larger problems the computational benefit will increase for the same level of difference between the tree method and the UVLM. In this case, a cutoff of 8 ring lengths and 200 spanwise panels leads to a speedup of 4.5 and 3.9 for the filament and double-tree methods, respectively, each of which also exhibit <1% difference in the force distribution and lift coefficient compared with the UVLM.

Taking into account the results from Section 6.4, a cutoff of eight ring lengths should reliably lead to an analysis with <5% difference with the basic UVLM for at least 20 time steps. This cutoff distance is shown to provide good speedup for large problems while keeping the solution reasonably accurate (compared with the UVLM).

6.6 | Parallelization scaling

Parallelization of the code requires special consideration because of the way the double-tree method divides up the domain. If there are P cores, ideally there should be P jobs, with each job being assigned $\frac{1}{P}$ of the domain. Since the

double-tree method divides the domain into cells, each job must be assigned to a cell in order to parallelize the algorithm. To do this, each job is assigned a different query tree cell to start the algorithm (instead of starting from the root cell). A problem arises, however, when the number of cores does not match any combination of cells that cover the entire domain. The proposed solution is to leave the "remainder" cores unused, which provides a balance between parallel scaling and eliminating unnecessary overhead.

For example, if every cell has four children and there are eight cores, first the children of the root cell are considered as the parallel jobs. Since at this point there are more cores than jobs, one of the cells is replaced with its children, bringing the total number of jobs to 7. Replacing another of the cells with its children increases the number of jobs to 10. With seven jobs and eight cores, the computation time will be reduced to $\frac{1}{7}$. With 10 jobs and 8 cores, the first 8 jobs will complete in $\frac{1}{10}$ time; however, the remaining 2 jobs are run afterwards, also in $\frac{1}{10}$ time, bringing the total time to $\frac{2}{10}$. Since $\frac{1}{7} < \frac{1}{5}$, it is better to leave 1 core unused than it is to have 2 extra jobs in this case. Furthermore, the creation of each job has an associated overhead cost, meaning that using a larger number of jobs than is necessary is detrimental to the overall computation time.

Since the number of children a cell can have does not change, the number of remainder cores does not exceed 2. This means that as the number of cores increases, the ratio of remainder cores to used cores will decrease, meaning that parallel scaling is preserved in this way.

Since the parallelized analysis distributes query tree cells among the cores, if the number of cores matches the number of leaves in the query tree, then each core is assigned a leaf cell with which to start its double-tree solution procedure search. This means that any computations that would be saved by rootward query tree cells are lost, moving the order

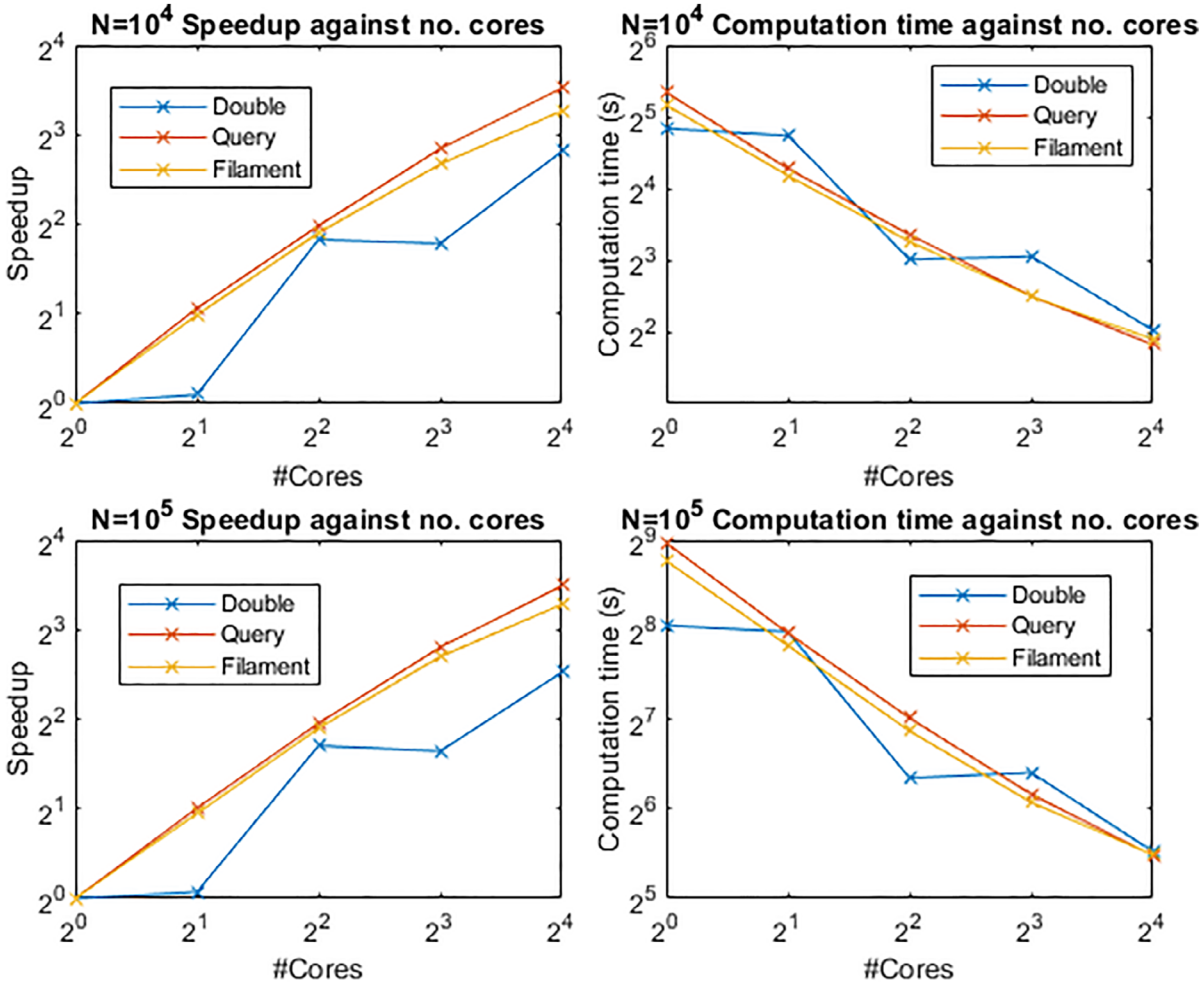

FIGURE 18 Base-2 log graphs of speedup and computation time for each tree method for different analysis sizes [Colour figure can be viewed at wileyonlinelibrary.com]

of computations toward $O(n \cdot \log n)$. Furthermore, any additional cores added beyond the number of leaves cannot have a job assigned to them, and so are wasted. For these reasons, an individual tree method may have better parallel scaling than the double tree if a large amount of parallelization is used relative to the size of the problem.

In order to measure how each method scales as more parallel cores are introduced, the same setup as the *order of magnitude* analysis (Section 6.1) is used with different numbers of panels ($N = 10^4$ and 10^5). Again, the results are averaged across three runs, each of which has the same trend, as shown in Figure 18. The analysis is run on a high-performance computing cluster on 2^x cores; $x = [0, 1, 2, 3, 4]$.

The speedup is calculated as the ratio of the 1-core computation time over the x-core computation time. It is immediately obvious that the double-tree parallelization scheme developed is sensitive to the number of cores available. For 2^1 and 2^3 cores, it is likely that the parallelization method was unable to reduce the size of the largest job compared with 2^0 and 2^2 cores respectively, thus the computation time is similar. These trends are the same for $N \approx 10^3$. At 10^2 panels and below, the analysis is too small to gain benefit from parallelization.

Given an improved parallelization scheme, the double-tree method has the potential to be faster than the other tree methods for any number of cores. Although the double-tree method has the worst speedup out of the three methods, it remains competitive for larger problems due to its smaller order of computations.

7 | CONCLUSIONS

The double-tree method provides a substantial reduction in the computational cost of the UVLM. The individual tree methods (filament and query point) are shown to be $O(n \cdot \log n)$ on their own, however when combined into the double-tree method, solve the problem in $O(n)$ time. All tree methods provide a similar level of cost reduction for the test cases investigated; however, the double-tree method performs better in terms of computational efficiency for larger problems.

The trade-off between accuracy and computational efficiency can be controlled by the cutoff distance. For the AR 40 test case with a cutoff distance of four ring lengths, the double-tree method achieves <5% difference in both force distribution and lift coefficient, while decreasing the total computation time by a factor of 7. $A < 1\%$ difference is achieved at a cutoff of eight ring lengths with a computational cost reduction factor of 4. Wake truncation is a simpler method to reduce the computational cost, however it requires about a 10-chord length cutoff for $a < 1\%$ difference in the solution.[13] Furthermore, no computational time is saved for the non-truncated panels, whereas the double-tree method reduces computations for all interactions beyond the cutoff distance, including wing-wing interactions. Finally, the double-tree method can easily be combined with wake truncation to further reduce computational cost, although further studies are required to understand the effect of the combined approach on the trade-off between accuracy and efficiency.

Results show that more complex geometry can cause a larger solution difference between the tree methods and the UVLM for the same cutoff distance; however, this difference still follows a predictable pattern within the same problem.

With an increase in problem size (number of panels), the speedup is shown to be more sensitive than the solution difference, meaning that larger problems will benefit from more speedup for the same level of accuracy. This means that applications of the UVLM where the computational cost is high, such as large-scale full-aircraft analysis in optimization and dynamic load case analysis, will benefit the most from the double-tree method.

The current double-tree parallelization algorithm is sensitive to the number of cores and has the worst speedup; however, it remains competitive in terms of computational cost for large parallelized problems.

8 | RECOMMENDATIONS FOR FURTHER WORK

Different agglomeration methods may be directly compared with the double-tree method, such as the vortex particle multipole moment method developed by Willis et al.[14] Other methods may provide a better overall performance than the UVLM-based approach, with the downside of being potentially more difficult to implement.

An improved parallelization scheme could be developed that divides the domain using both trees simultaneously, thus reducing the initial tree depth used by the parallel jobs. This could save computations by allowing for shallower portions of each tree to double-agglomerate. It could also raise the upper limit on the number of usable cores for parallelization.

In the method presented in this paper, the cutoff distance is a constant value throughout the simulation. However, it may be beneficial to develop an adaptive cutoff distance method, in order to improve the algorithm's performance. Investigations into how the cutoff distance should be chosen for a range of flight configurations would also be beneficial.

ACKNOWLEDGEMENTS

Some results in this work were obtained using the Maxwell High Performance Computing Cluster of the University of Aberdeen IT Service (www.abdn.ac.uk/staffnet/research/hpc.php), provided by Dell Inc. and supported by Alces Software. The lead author would also like to thank the University of Aberdeen for their research scholarship funding.

ENDNOTE

[1] The shedding ratio determines the location of the vortex filaments on the trailing edge of the wing according to a fraction of the edge's displacement during the last timestep.[1]

ORCID


Bryn Jones https://orcid.org/0000-0002-8680-2738

REFERENCES

1. Katz J, Plotkin A. Lifting-surface solution by vortex ring elements. *Low Speed Aerodynamics*. 2nd ed. New York, NY: Cambridge University Press; 2001:419-429.
2. Murua J, Palacios R, Graham JMR. Applications of the unsteady vortex-lattice method in aircraft aeroelasticity and flight dynamics. *Prog Aerosp Sci*. 2012;55:46-72. https://doi.org/10.1016/j.paerosci.2012.06.001.
3. Wei X, Ng BF, Zhao X. Aeroelastic load control of large and flexible wind turbines through mechanically driven flaps. *J Franklin Inst*. 2019;356(14):7810-7835. https://doi.org/10.1016/j.jfranklin.2019.02.030.
4. Simoes F, Graham J. Application of a free vortex wake model to a horizontal axis wind turbine. *J Wind Eng Ind Aerodyn*. 1992;39(1–3):129-138. https://doi.org/10.1016/0167-6105(92)90539-M.
5. Fritz TE, Long LN. Object-oriented unsteady vortex lattice method for flapping flight. *J Aircr*. 2004;41(6):1275-1290. https://doi.org/10.2514/1.7357.
6. Ghommem M, Collier N, Niemi AH, Calo VM. On the shape optimization of flapping wings and their performance analysis. *Aerosp Sci Technol*. 2014;32(1):274-292. https://doi.org/10.1016/j.ast.2013.10.010.
7. Verstraete ML, Preidikman S, Roccia BA, Mook DT. A numerical model to study the nonlinear and unsteady aerodynamics of bioinspired morphing-wing concepts. *Int J Micro Air Veh*. 2015;7(3):327-345. https://doi.org/10.1260/1756-8293.7.3.327.
8. Ghommem M, Hajj MR, Mook DT, et al. Global optimization of actively morphing flapping wings. *J Fluids Struct*. 2012;33:210-228. https://doi.org/10.1016/j.jfluidstructs.2012.04.013.
9. Abdelkefi A, Ghommem M. Piezoelectric energy harvesting from morphing wing motions for micro air vehicles. *Theor Appl Mech Lett*. 2013;3(5):052004. https://doi.org/10.1063/2.1305204.
10. Wang Z, Chen PC, Liu DD. Nonlinear-aerodynamics/nonlinear-structure interaction methodology for a high-altitude long-endurance wing. *J Aircr*. 2010;47(2):556-566. https://doi.org/10.2514/1.45694.
11. Murua J, Palacios R, Graham JMR. Assessment of wake-tail interference effects on the dynamics of flexible aircraft. *AIAA J*. 2012;50(7):1575-1585. https://doi.org/10.2514/1.J051543.
12. Werter NPM, De Breuker R. A novel dynamic aeroelastic framework for aeroelastic tailoring and structural optimisation. *Compos Struct*. 2016;158:369-386. https://doi.org/10.1016/j.compstruct.2016.09.044.
13. Werter NP, Breuker RD, Abdalla MM. Continuous-time state-space unsteady aerodynamic modelling for efficient aeroelastic load analysis. Paper presented at: International forum on aeroelasticity and structural dynamics; Saint Petersburg, Russia; 2015. https://www.semanticscholar.org/paper/Continuous-time-state-space-unsteady-aerodynamic-Werter-Breuker/1af4063e79383e26d051aeb2dc158e1ca0b16816. Accessed June 14, 2019.
14. Willis DJ, Peraire J, White JK. A combined pFFT-multipole tree code, unsteady panel method with vortex particle wakes. *Int J Numer Methods Fluids*. 2006;53(8):1195-1422. https://doi.org/10.1002/fld.1240.
15. Wales CJA, Valente C, Cook RG, Gaitonde AL, Jones DP, Cooper JE. The future of non-linear modelling of aeroelastic gust interaction. Paper presented at: AIAA Aviation Forum. Atlanta, GA; 25-29 June 2018. doi:10.2514/6.2018‐3632
16. Kebbie-Anthony A, Gumerov NA, Preidikman S, Balachandran B, Azarm S. Fast multipole method for nonlinear, unsteady aerodynamic simulations. Paper presented at: AIAA Modeling and Simulation Technologies Conference. Kissimmee, FL; 8-12 January 2018. doi:10.2514/6.2018‐1929
17. Kebbie-Anthony A, Gumerov NA, Preidikman S, Balachandran B, Azarm S. Fast multipole accelerated unsteady vortex lattice method based computations. *J Aerosp Inf Syst*. 2019;16(6):237-248. https://doi.org/10.2514/1.I010690.

18. Gumerov NA, Duraiswami R. Efficient FMM accelerated vortex methods in three dimensions via the Lamb–Helmholtz decomposition. *J Comput Phys*. 2013;240:310-328. https://doi.org/10.1016/j.jcp.2013.01.021.
19. Bertin JJ, Smith ML. *Aerodynamics for Engineers*. Vol 294. 3rd ed. New Jersey: Prentice Hall; 1998.
20. Vassberg J, DeHaan M, Rivers S, Wahls R. Development of a common research model for applied CFD validation studies. Paper presented at: 26th AIAA Applied Aerodynamics Conference. Honolulu; 18-21 August 2008. doi:10.2514/6.2008‐6919

How to cite this article: Jones B, Dunning P, Maheri A. The double-tree method: An $O(n)$ unsteady aerodynamic lifting surface method. *Int J Numer Meth Fluids*. 2020;92:1394–1414. https://doi.org/10.1002/fld.4833

APPENDIX A. ANALYSIS SIZE TABLE DATA

TABLE 1

Number of spanwise panels	Tree	Cutoff dist. (ring lengths)	Force dist. max diff. (%)	Lift coeff. max diff. (%)	$\frac{t^{\text{UVLM}}}{t^{\text{tree}}}$
200	Filament	1	73.85	−20.36	20.180
	Query		24.39	12.40	15.938
	Double		20.79	−40.92	20.700
100	Filament		50.01	−19.73	12.008
	Query		24.39	11.87	8.8445
	Double		20.27	−39.86	10.835
50	Filament		32.36	−18.27	6.4236
	Query		24.44	10.95	4.9271
	Double		19.65	−38.63	5.6960
25	Filament		18.41	−15.98	3.4419
	Query		24.69	9.75	2.6189
	Double		17.58	−32.72	2.6642
200	Filament	2	35.62	−9.74	14.335
	Query		12.77	6.18	10.670
	Double		10.33	−10.82	13.000
100	Filament		23.67	−9.10	7.6584
	Query		12.96	5.76	5.5812
	Double		10.59	−10.80	6.5467
50	Filament		15.63	−8.09	4.1613
	Query		13.28	5.08	3.0908
	Double		10.37	−10.34	3.4718
25	Filament		12.28	−6.30	2.3336
	Query		13.53	4.57	1.7331
	Double		10.64	−7.66	1.7795
200	Filament	4	13.66	−3.87	8.1785
	Query		6.63	4.65	5.8721
	Double		3.77	−4.18	7.0798

(Continues)

TABLE 1 (Continued)

Number of spanwise panels	Tree	Cutoff dist. (ring lengths)	Force dist. max diff. (%)	Lift coeff. max diff. (%)	$\frac{t^{\text{UVLM}}}{t^{\text{tree}}}$
100	Filament		9.01	−3.50	4.3717
	Query		7.27	−5.08	3.1471
	Double		3.61	−3.95	3.7214
50	Filament		5.35	−2.86	2.3586
	Query		8.04	−6.31	1.7197
	Double		3.44	−3.65	1.9534
25	Filament		2.93	−1.95	1.3371
	Query		7.73	−5.95	0.9795
	Double		3.33	−3.22	1.0437
200	Filament	8	0.91	−0.99	4.5411
	Query		1.25	1.46	3.2633
	Double		0.70	−0.83	3.9453
100	Filament		0.75	−0.81	2.4014
	Query		1.17	1.29	1.7192
	Double		0.66	−0.77	2.0590
50	Filament		0.48	−0.54	1.3124
	Query		1.06	1.03	0.9424
	Double		0.52	−0.59	1.0899
25	Filament		0.21	−0.24	0.7643
	Query		0.89	0.71	0.5522
	Double		0.21	−0.21	0.6161