# Consistent Multi-Threaded Mesh Refinement with Adaptive Length Scale Estimation for Moving Boundary Problems

Chao Li
State Key Laboratory of High Performance Computing, College of Computer, National University of Defense Technology, Changsha, China
dirk911@nudt.edu.cn

Ran Zhao
State Key Laboratory of High Performance Computing, College of Computer, National University of Defense Technology, Changsha, China
zhaoran13@nudt.edu.cn

Xiaowei Guo
State Key Laboratory of High Performance Computing, College of Computer, National University of Defense Technology, Changsha, China
guoxiaowei@nudt.edu.cn

## ABSTRACT

Computational fluid dynamic simulations with moving boundaries are widely involved in high performance computing applications. For problems with large-displacement or large-deformation boundaries, mesh cells near the boundaries are often excessively stretched or compressed, thus it's hard to maintain a high-quality mesh. To deal with the distorted cells, this paper adopts the mesh refinement method based on the open source software OpenFOAM. In order to achieve the desired effects of localization and adaptation, we propose an adaptive length scale estimation algorithm based on the specified growth factor and current edge lengths. Considering the inconsistency problems for the original implementation of parallelization, an optimized multi-threaded master/worker model is developed for the process of edge checking. Experiments show that our adaptive length scale estimation algorithm works well for moving boundary problems. Compared to the original mesh deformation, using the adaptive mesh refinement could greatly improve the mesh quality. In parallel testing, all the results are consistent and a maximum speedup of 3.8 is achieved on a computing node of 24 cores.

## CCS CONCEPTS

• **Applied computing**; • **Physical sciences and engineering**; • **Engineering**; • **Computer-aided design**; • **Computing methodologies**; • **Parallel computing methodologies**; • **Parallel algorithms.**;

## KEYWORDS

Moving boundary, Mesh refinement, Local adaptation, Length scale estimation, Consistent thread parallelization

**ACM Reference Format:**
Chao Li, Ran Zhao, and Xiaowei Guo. 2021. Consistent Multi-Threaded Mesh Refinement with Adaptive Length Scale Estimation for Moving Boundary Problems. In *The 5th International Conference on Computer Science and Application Engineering (CSAE 2021), October 19–21, 2021, Sanya, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3487075.3487104


*CSAE 2021, October 19–21, 2021, Sanya, China*

ACM ISBN 978-1-4503-8985-3/21/10…$15.00
https://doi.org/10.1145/3487075.3487104

## 1 INTRODUCTION

Simulation of Computational Fluid Dynamics (CFD) with moving boundary is widely involved in high performance computing applications, such as aerospace engineering and ocean engineering [1-3]. Mesh deformation is an effective method to address moving-boundary problems. To deal with different moving conditions, researchers have developed a lot of mesh deformation methods [4-7].

The unified point of all the mesh deformation methods is to calculate the mesh motion by a specific algorithm while keeping the mesh topology unchanged. However, for problems with large-displacement or large-deformation boundaries, deformation methods could hardly maintain high-quality meshes. Because the topology could not be modified, mesh cells near the boundaries will be excessively stretched or compressed. Adaptive mesh reconnection is often used in situations where cells have become excessively distorted. One approach is to re-mesh the whole domain using the mesh regeneration algorithm. But this method produces too much time cost and introduces excessive interpolation errors. Therefore, a more applicable approach is the local re-meshing with less time consuming and interpolation errors [8].

The topic of local re-meshing could be extended to include refinement and de-refinement of cells. The key idea is to increase the points at regions which require higher mesh density to resolve high solution gradients and decrease the mesh density where the solution error is low [9-11]. Edge bisection and edge contraction are two opposite operations that could refine and coarsen the mesh resolution. These operations could be used together to modify the mesh topology and improve the mesh quality [8] [12].

OpenFOAM is the most widely used open-source software in CFD [13] [14]. This paper adopts the local refinement method implemented in the 4th extended version of OpenFOAM [15]. However, in our practical cases with moving boundaries, several technological problems have arisen for the original implementation.

Firstly, the original algorithm needs to specify a fixed-value length scale at the boundary. However, for a new-generated initial mesh, we often do not know the exact lengths for the boundary cells. In addition, the lengths of the boundary cells may not keep the same, so it's not reasonable to define a specific value for all the cells on the boundary. Secondly, only one specified growth factor is utilized to calculate the length scale of the internal mesh. This will lead to an incongruous change of the mesh resolution. The length

scale should be calculated according to the initial mesh. Thirdly, the thread parallelization of the original implementation has a inconsistency problem. Under the same configuration condition, the results of multiple runs are different.

To address the above problems, this paper proposes an adaptive length scale estimation algorithm based on the specified growth factor and current edge lengths. Considering the inconsistency problems for the original implementation, an optimized multi-threaded master/worker model is developed for the parallelization of edge checking. The parallelization is thread based so it is shared memory parallelization and not scalable on distributed servers. The concrete content is organized as follows. Section 2 introduces the basic operations for the mesh refinement. Section 3 elaborates the details for the adaptive length scale estimation. The optimized thread parallelization is involved in Section 4. The experiments and conclusions are respectively contained in Section 5 and 6.

## 2 MESH REFINEMENT BASIS

### 2.1 Edge Bisection and Contraction

For numerical simulation with moving boundary, the flow field in the computational area sometimes changes dramatically due to the movement of the boundary. If using a sparse uniform mesh, it is hard to ensure the calculation accuracy in these regions, thus the details of the fluid will not be captured. If a dense uniform mesh is used, large amounts of computation will be produced. Therefore, we can use non-uniform mesh to address this problem: using sparse mesh in the areas of laminar flow and using dense mesh in the areas with large-gradient changed flow. For steady flows, we can refine the initial static mesh locally according to experience, while for unsteady flows with moving boundaries, it is necessary to refine and coarsen the mesh during the running time according to the changes of flow field. In this paper, edge bisection and edge contraction are used to adjust the density of the mesh.

Edge bisection splits an edge at the midpoint [16]. Then all the edges connected to the original edge will be separated and new edges and cells will be generated. Figure 1 shows the operation of edge bisection. As we can see, A-B is an edge shared by two triangular cells. Due to its length is relatively large, a new point C is generated at the midpoint and then is connected to the other two points. Thus, the two mesh cells sharing A-B are bisected into four new cells.

Edge contraction is the inverse operation of edge bisection [17]. It firstly removes one endpoint of an edge from the mesh, and then deletes all the edges that connected with this endpoint. The remaining point that originally connected with this endpoint is then reconnected to the other endpoint. In Figure 1, edge C-D is relatively too short in cell A-C-D and cell B-C-D. So, point D is deleted and C-D is compressed into point C.

### 2.2 Checking by Length Scale

In the process of mesh density adjustment, it should be firstly figured out which regions need to be refined or coarsened. We need to check the length of each edge to perform a proper operation on it. This could be regarded as a classification problem and we can divide the mesh edges into three types: subset that needs to be bisected, subset that needs to be contracted, and subset that doesn't need to be adjusted. In this paper, we classify the mesh edges based on the length scales.

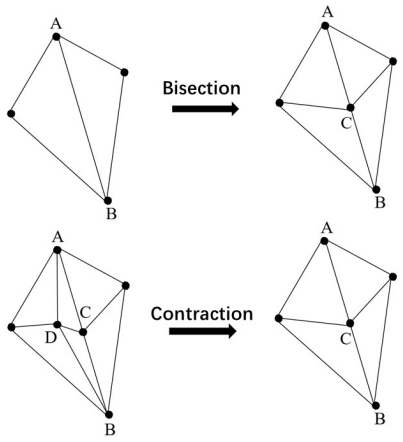


**Figure 1: Edge Bisection and Contraction for Mesh Density Adjustment.**

According to Menon [15], length scale $L(x)$ is defined as the size range of cell $x$. For an edge $e$ shared by cells $owner[e]$ and $neighbor[e]$, the classification could be accomplished as follows:

$$e \in \begin{cases} Subset_{Bisct}, Length(e) > F(e) \times Max \\ Subset_{Ctrct}, Length(e) < F(e) \times Min \\ Subset_{Non}, F(e) \times Min \leq Length(e) \leq F(e) \times Max \end{cases} \quad (1)$$

where $F(e) = \frac{L(owner[e])+L(neighbor[e])}{2}$, which represents the standard length scale for edge $e$. $Max$ and $Min$ are predefined factors for the scaling ratio. Different configuration values lead to different mesh qualities and they are specified to 1.4 and 0.7 for the case of moving fish in this paper. Only edge lengths within the length scale qualified by $L(e)$, the topology will not be modified. Otherwise, the edges will be bisected or contracted.

## 3 ADAPTIVE LENGTH SCALE ESTIMATION

This paper estimates the length scale based on a greedy method proposed by Menon [18]. The method takes the specified length scale of boundary cell as the basis and increases/decreases the length scales of internal cells according to a specified growth factor. However, in actual simulations, the initial static meshes are generated by users according to the practical physical conditions. It is hard to manually specify the boundary scale and growth factor to be consistent with the mesh. The initial distribution of the mesh density is commonly the most direct response to the user's intention and is often complex. So, another problem is that for an initial mesh generated with a non-uniform density distribution, if the length scale is estimated only by a predefined growth factor, the resolution of the mesh will change in a way vastly different from the distribution of the initial mesh.

Since the initial static mesh directly represents the physical characteristics, if the mesh density distribution is consistent with the initial state all along during the boundary motion, an ideal simulation condition will be achieved. Considering this, we add a

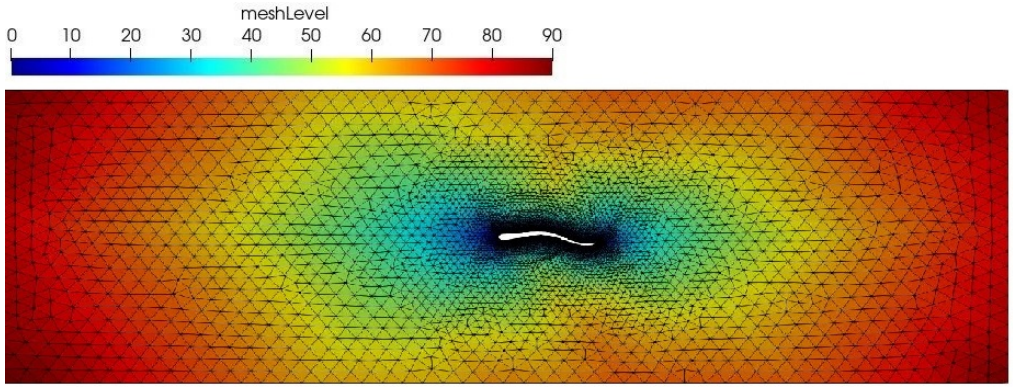


**Figure 2: Level Distribution of the Initial Mesh.**

third item to Menon's conclusion, thus a reasonable length scale should have following three properties:

- For mesh cells on the moving boundary $x \in \delta\Omega$, $L(x)$ should be small enough to capture the curvature details with sufficient accuracy.
- For mesh cells defined in the inner domain, $L(x)$ should grow from $\delta\Omega$ to the interior according to a certain gradient, so as to reduce the computational cost of the solution.
- For all mesh cells in the solution domain $x \in \Omega$, $L(x)$ should be consistent with the initial mesh's length distribution, so as to avoid excessive changes.

In order to meet the above requirements, we develop an adaptive length scale estimation algorithm based on the specified growth factor and current edge lengths, as shown in Algorithm 1. Firstly, all the mesh cells are traversed in step 1-5, and the average side lengths are calculated and assigned to $L(x)$. Because the length scales are initialized based on the side lengths of the initial mesh cells, this assures the values of the length scales have obtained the initial information and could maintain the same distribution as the initial mesh.

In step 6-32, the length scales are further modified by the specified growth factor based on the idea of hierarchical advancing. Here hierarchical advancing means that we take the moving boundary as the center and divide the mesh cells into different levels. As shown in Figure 2, the closer the cell is to the center, the lower its level is. Then, from low-level cells to high-level cells, the length scales are increased according to the specified growth factor $\alpha$ so as to meet the above properties. Specifically, we firstly initialize the cells' levels to 0 at step 4, and then set the levels of moving boundary cells to 1 in step 6-11. For mesh cells at level 1, their length scales remain static during the whole simulation, which ensures the scales near the moving boundary could be consistent with the initial state. In step 12-32, we use the *while* loop to advance the mesh hierarchy computation. For the current visiting level *curLevel*, we firstly find all the neighbors of each cell $x$ in *curLevelCells*. For each neighbor $y$ of cell $x$, if its level is 0, which means it has never been visited, then we set its level to current-visiting-level plus 1. Then we calculate the average length scale *avgScale* of $y$'s neighbors whose levels are lower than it. If the current length scale $L(y)$ is greater than $avgScale \times \alpha$ or less than $avgScale \div \alpha$, which means the changing gradient of current length scale is greater than the predefined growth factor $\alpha$,

$$L(y) should be reset to these values. Then cell$$

is added to the stack of next visiting level. After traversing all the cells of current visiting level, the next level will be pushed forward. The process is ended when all the mesh cells have been accessed.

**Algorithm 1** Adaptive length scale estimation.

**Require:** mesh $M$, index of mesh cells $cells(M)$, index of boundary cells $boundaryCells(M)$, growth factor $\alpha$
**Ensure:** length scales $L(M)$
1: **for all** $x \in cells(M)$ **do**
2: Calculate the average side length of mesh cell x :$avgLength$
3: Set the length scale of x to the average side length: $L(x) = avgLength$
4: Initialize the level of x: $cellLevels(x) = 0$
5: **end for**
6: Initialize the current visiting level: $curLevel = 1$, and the current visited count: $visitedCells = 0$
7: **for all** $x \in boundaryCells(M)$ **do**
8: Set the level of boundary cell to 1: $cellLevels(x) = curLevel$
9: Insert x to the stack of current level: $levelCells.insert(x)$
10: Increase the current visited count: $visitedCells$++
11: **end for**
12: **while** $visitedCells < sizeof(M)$ **do**
13: Get the current level cells: $curLevelCells = levelCells.toc()$
14: Clear the stack of current level: $levelCells.clear()$
15: **for all** $x \in curLevelCells$ **do**
16: Find the neighbor cells of x: $neighbor(x)$
17: **for all** $y \in neighbor(x)$ **do**
18: **if** $cellLevels(y) == 0$ **then**
19: Set the level of y to the next level: $cellLevels(y) = curLevel + 1$
20: Calculate the average length scale $avgScale$ of $y$'s neighbors whose levels are lower than $y$;
21: **if** $L(y) > avgScale \times \alpha$ **then**
22: Reset the length scale: $L(y) = avgScale \times \alpha$
23: **else if** $L(y) < avgScale \div \alpha$ **then**
24: Reset the length scale: $L(y) = avgScale \div \alpha$
25: **end if**
26: Insert y to the stack of next level: $levelCells.insert(y)$
27: Increase the current visited count: $visitedCells$++
28: **end if**
29: **end for**
30: **end for**
31: Increase the current visiting level: $curLevel$++.
32: **end while**

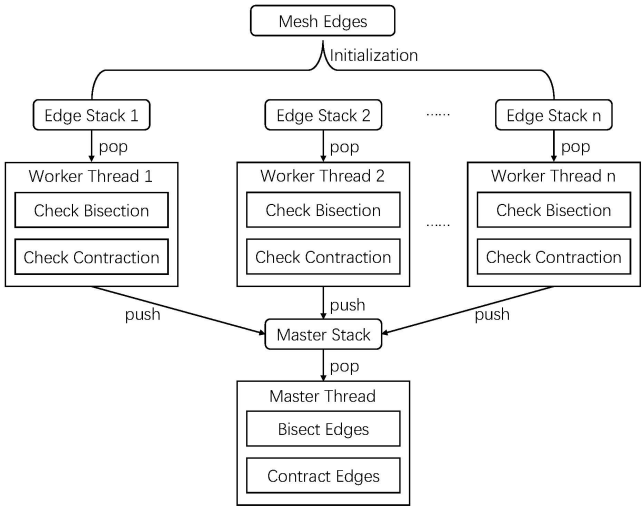


**Figure 3: Master/Worker Threading Model.**

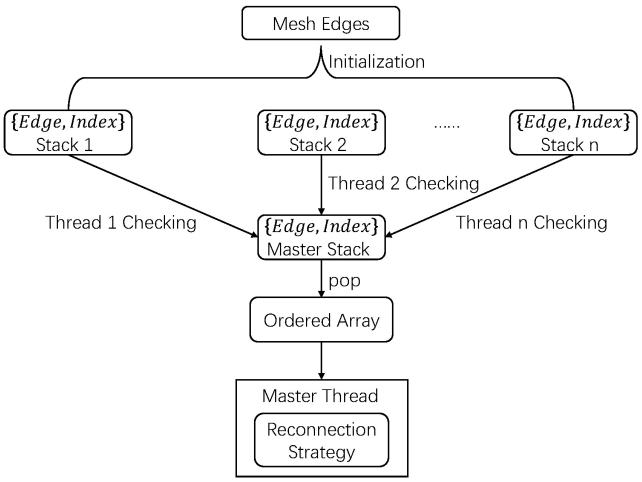


**Figure 4: Optimized Threading Model.**

The key point of Algorithm 1 is that the length scale could be adjusted according to the current lengths of cell edges at running time and is also refined by a reasonable growth factor in case of too large changing gradient. Based on a lot of experiments, we find that a reasonable growth factor should be set to around 1.1. Applied with our algorithm, the mesh edges could be adaptively bisected and contracted, and the mesh topology could always keep the same scale distribution as the initial static mesh.

# 4 CONSISTENT THREAD PARALLELIZATION

## 4.1 Master/Worker Threading Model

In OpenFOAM-Extend 4.0, the above mesh refinement is accomplished based on a master/worker threading model. To address the locking problem for thread parallelization, the model is implemented by a coarse-grained locking mechanism, which performs the checking in parallel but sequentially performs each operation.

As shown in Figure 3, according to the number of threads, the mesh edges are equally divided into $n$ stacks. Each worker thread possesses a unique stack. In the process of the checking, each thread traverses its own stack and judges whether the edges need to be modified. Detailed implementation of the edge checking could refer to [12]. If an edge is checked to be bisected or contracted, it will be pushed into the shared master stack. All the stack operation *push* and *pop* are protected by the locking mechanism. After all the worker threads have finished the checking, a synchronization will be performed to kill the worker threads. At last, the master thread bisects or contracts all the edges in the master stack, and the mesh refinement at this time step is then finished.

## 4.2 Inconsistency Problem and Optimization

Consistency is a basic property for actual simulations. Under a certain configuration without randomness, any simulation results should be the same and could be replicable. However, in the experiments we find the above parallel model has a inconsistency problem, which leads to different refinement results under a certain condition.

In the threading model, all the worker threads are sharing a unique master stack. Because the master stack is protected by the locking mechanism, only one thread could access the stack at a time and the other threads have to waiting until the *push* operation is finished. When the master stack is released, the current waiting threads will access it in an uncertain sequence scheduled by the operating system. Thus, it will result in an uncertain stack for the master thread to handle. Although the mesh edges are initialized into $n$ certain and isometric arrays, the generated master stack is in a changeable order, which then leads to the inconsistency problem.

From the above analysis, we could find the key point is the changeable and uncertain order of the edges in the master stack. If we could generate a certain array for the master thread to handle, the results will be unchangeable. Based on this, we attach a unique index for each edge and make them as a coupled set:

$$\{Edge,\ Index\}\,.$$

The index for each edge is defined according to the mesh levels. Smaller level corresponds to smaller index. Then the coupled sets are treated as the operating object. As shown in Figure 4, different from the original model, all the stacks store the sets instead of the edges. Although the master stack is still uncertain, we could rearrange the edges according to their indexes and then generate an ordered edge array. This ordered array is unique without the influence of multiple threads. Thus, the mesh refinement results could be consistent in parallel.

## 4.3 Reconnection Strategy

In the optimized threading model, when the master thread gets the ordered edge array and starts to reconnect the checked edges, a problem arises that whether bisection or contraction should be performed at first step. In this paper, we conclude three reconnection strategies, as shown in Figure 5

To classify whether the edges are checked to be bisected or contracted, we construct two ordered arrays ‘arrayB’ and ‘arrayC’ from

```
B-C
forAll(arrayB, i)
{
  if(bisectCheck(arrayB[i]))
    bisectEdge(arrayB[i]);
}
forAll(arrayC,i)
{
  if(contractCheck(arrayC[i]))
    contractEdge(arrayC[i]);
}
```

```
C-B
forAll(arrayC, i)
{
  if(contractCheck(arrayC[i]))
    contractEdge(arrayC[i]);
}
forAll(arrayB,i)
{
  if(bisectCheck(arrayB[i]))
    bisectEdge(arrayB[i]);
}
```

```
Mix
forAll(array, i)
{
  if(bisectCheck(array[i]))
    bisectEdge(array[i]);
  else
  if(contractCheck(array[i]))
    contractEdge(array[i]);
}
```

**Figure 5: Reconnection Strategies.**

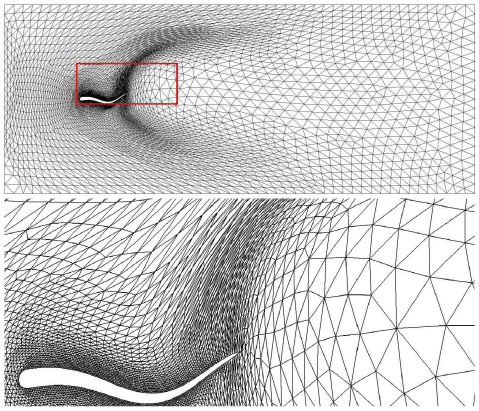

**Figure 6: Mesh Deformation without Topology Refinement.**

the master stack. The strategy 'B-C' indicates that bisections are firstly performed and then contractions are followed. For 'B-C', 'arrayB' is firstly traversed and 'arrayC' is followed. While for strategy 'C-B', bisections are performed after all the contractions have finished. So, 'arrayC' is firstly traversed and 'arrayB' is followed. For strategy 'Mix', all the checked edges are stored in a single ordered array, and bisections and contractions are performed alternately in a certain order. Here checks are performed again to classify the operation type for each edge. It should be noted that, because the front operations(either bisection or contraction) could influence the necessity of the latter operations, checks are also performed in strategy 'B-C' and 'C-B'.

# 5 EXPERIMENT AND DISCUSSION

## 5.1 Experiments Setup

In this paper, we conduct experiments in a computing cluster with two Intel Xeon Gold 5118 12-cores CPU for each node. Parallelization between multiple nodes is not considered in this work. So, all the test cases are simulated within one node.

The benchmark is a two-dimensional fish with length of $L$ located in the center of a rectangle with mesh of 10339 cells, as shown in Figure 2. The governing equation of the fish motion is from Carling [19] as following:

$$y(s,t) = 0.125\frac{s+0.03125}{1.03125}\sin\left[2\pi\left(s-\frac{t}{T}\right)\right] \tag{2}$$

where $y$ is the displacement of mid-line profile, $s$ is the arc length along the mid-line of the fish, $t$ is the current time and $T$ is the undulating period. The fish is moving ahead at a speed of $0.5L/T$.

**Table 1: Mesh Quality Information at Different Time Steps**

| Time Step | Lowest Quality | Average Quality | Cell Number |
|---|---|---|---|
| $t = 0T$ | 0.3671 | 0.9528 | 10339 |
| $t = 2.2T$ | 0.2909 | 0.8944 | 10929 |
| $t = 4.5T$ | 0.2723 | 0.8700 | 10922 |
| $t = 6.8T$ | 0.2892 | 0.8627 | 11152 |
| $t = 8T$ | 0.2644 | 0.8643 | 10815 |

## 5.2 Effectiveness of Adaptive Refinement

We firstly test the mesh motion without topology refinement. The result at $t$=6.8$T$ is illustrated in Figure 6. We could see the topology of the whole mesh remains unchanged and only the locations of the points are modified. From the detailed view around the fish body, it can be seen many cells have been excessively stretched or compressed. Commonly these low-quality cells could not support the numerical simulation any more. Thus, local mesh refinement is required.

The mesh refinement result with adaptive length scale estimation is shown in Figure 7. For all the tests, the growth factor is specified to 1.1 and the reconnection strategy is set to 'Mix'. Four specific time steps are exhibited to show different morphologies of the fish motion. Different color represents different generated time for each cell. The time is reduced from red to blue and the dark red regions are the newly generated cells. We could see as the fish approaches to the left, the cells in front of the fish body will be refined and the cells behind the tail will be coarsened according to the growth factor. Thanks to the adaptive length scale estimation, during the whole process the mesh around the fish body has always kept in a same resolution as the initial state.

Table 1 shows the mesh quality information at these time steps. The quality metric is a scalar value which is normalized to vary in the range between 0 and 1, where a quality of 0 denotes a degenerate cell with zero volume, and 1 denotes a cell that is close to ideal. Specific formula can refer to [12]. In the table, we could see although the lowest quality and the average quality are reduced compared to the initial time, the variations are relatively small within an acceptable range. The number of cells also remains stable, which guarantees a consistent mesh scale as the initial state.

As a comparison to Figure 6, a detailed view around the fish body at $t$=6.8$T$ is illustrated in Figure 8. We could see there is no excessively stretched or compressed cell formed. Especially for the regions beside the tail, all the cells have kept in reasonable triangular shapes.

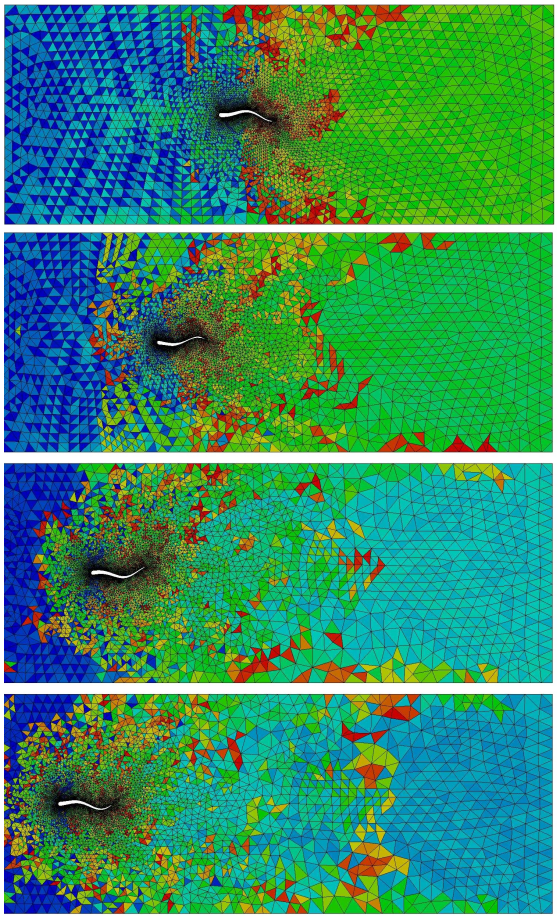

**Figure 7: Mesh Refinement with Adaptive Length Scale Estimation at Different Time Steps ($t$=2.2$T$, 4.5$T$, 6.8$T$, 8$T$).**

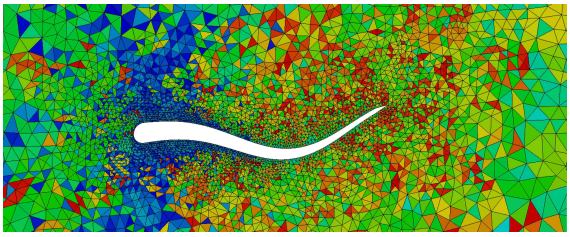

**Figure 8: Detailed View around the Fish Body at $t$=6.8$T$.**

## 5.3 Effectiveness of Consistent Parallelization

To give a visual display of the inconsistency problem, Figure 9 shows the variation of the cell number using the original algorithm. The black solid line is the result of serial running. It has shown a growing trend as the time proceeds, which means the mesh resolution is changing too much from the initial state. The dashed lines are four parallel results running with 4 cores. We could see they are definitely different from each other, although the configurations are unchanged. As a comparison, the test using the consistent algorithm

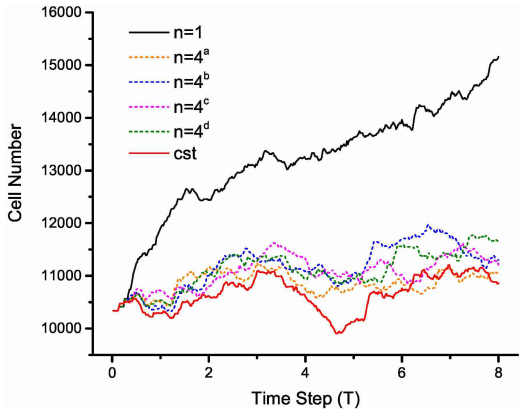


**Figure 9: Variation of the Cell Number Using the Original and Consistent Algorithms.**

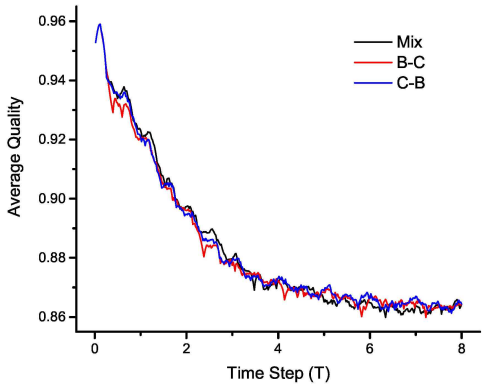


**Figure 10: Variation of the Average Quality Using the Three Different Connection Strategies.**

is also plotted in red solid line. The results are constant in both parallel and serial and the variation is much more stable than the non-consistent results.

Figure 10 shows the variation of the average quality using three different connection strategies. We could see there is not much difference between the three strategies in our test case. However, this does not mean the reconnection strategy has little impact on the mesh refinement. For large-scale cases with more bisections and contractions, the difference may be much more distinct.

The parallel speedup of the edge checking process is illustrated in Figure 11. At first, the speedup is increased with the growth of parallelism. Because the computing node has 24 physical cores, when the degree of parallelism reaches to 24, the parallel speedup starts to drop down. The maximum speedup is obtained at $n$=22 and is approximated to 3.8. This is because the node we used is also responsible for other system calls, so we could not occupy all the cores for computing. The parallel efficiency is restricted by the locking mechanism of the stack operations. Threads are frequently waiting for the access of the master stack.

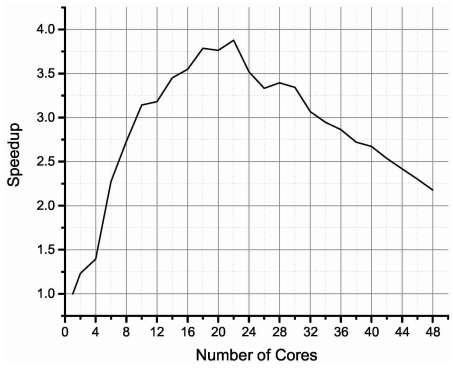


**Figure 11: Parallel Speedup of the Edge Checking Versus Degree of Parallelism.**

## 6 CONCLUSION

This paper proposes an adaptive length scale estimation for the local mesh refinement. It has been tested to be effective for the moving boundary problems. A consistent model is developed for the master/worker threading parallelization in the process of edge checking. The results show that no inconsistency problem exists in the optimized implementation. Some future work is on the way, such as more efficient locking mechanism, distributed parallelization, three-dimensional application, and so on.

## ACKNOWLEDGMENTS

The authors would like to thank the Natural Science Foundation of Hunan Province (Grant No. 2019JJ50723) and the open fund from the State Key Laboratory of High Performance Computing (No. 202001-04).

## REFERENCES

[1] Borazjani, I., & Sotiropoulos, F. (2008). Numerical investigation of the hydrodynamics of carangiform swimming in the transitional and inertial flow regimes. Journal of experimental biology, 211(10), 1541-1558.
[2] Jakobsson, S., & Amoignon, O. (2007). Mesh deformation using radial basis functions for gradient-based aerodynamic shape optimization. Computers & Fluids, 36(6), 1119-1136.
[3] Li, C., Yang, W., Xu, X., Wang, J., Wang, M., & Xu, L. (2017). Numerical investigation of fish exploiting vortices based on the Kármán gaiting model. Ocean Engineering, 140, 7-18.
[4] Rendall, T. C., & Allen, C. B. (2009). Efficient mesh motion using radial basis functions with data reduction algorithms. Journal of Computational Physics, 228(17), 6231-6249.
[5] Gao, X., Dong, Y., Xu, C., Xiong, M., Wang, Z., & Deng, X. (2017). Developing a new mesh deformation technique based on support vector machine. International Journal of Computational Fluid Dynamics, 31(4-5), 246-257.
[6] Li, C., Xu, X., Wang, J., Xu, L., Ye, S., & Yang, X. (2018). A parallel multiselection greedy method for the radial basis function–based mesh deformation. International Journal for Numerical Methods in Engineering, 113(10), 1561-1588.
[7] Zhao, R., Li, C., Guo, X., Fan, S., Wang, Y., & Yang, C. (2019). A block iteration with parallelization method for the greedy selection in radial basis functions based mesh deformation. Applied Sciences, 9(6), 1141.
[8] Dai, M., & Schmidt, D. P. (2005). Adaptive tetrahedral meshing in free-surface flow. Journal of computational Physics, 208(1), 228-252.
[9] Alauzet, F. (2013). Efficient moving mesh technique using generalized swapping. In Proceedings of the 21st International Meshing Roundtable (pp. 17-37). Springer, Berlin, Heidelberg.
[10] Alauzet, F. (2014). A changing-topology moving mesh technique for large displacements. Engineering with Computers, 30(2), 175-200.
[11] Baker, T., & Cavallo, P. (1999). Dynamic adaptation for deforming tetrahedral meshes. In 14th Computational Fluid Dynamics Conference (p. 3253).
[12] Menon, S. (2011). A numerical study of droplet formation and behavior using interface tracking methods.
[13] Jacobsen, N. G., Fuhrman, D. R., & Fredsøe, J. (2012). A wave generation toolbox for the open-source CFD library: OpenFoam®. International Journal for numerical methods in fluids, 70(9), 1073-1088.
[14] Guo, X. W., Xu, X. H., Wang, Q., Li, H., Ren, X. G., Xu, L., & Yang, X. J. (2016, May). A hybrid decomposition parallel algorithm for multi-scale simulation of viscoelastic fluids. In 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS) (pp. 443-452). IEEE.
[15] Menon, S., Mooney, K. G., Stapf, K. G., & Schmidt, D. P. (2015). Parallel adaptive simplical re-meshing for deforming domain CFD computations. Journal of Computational Physics, 298, 62-78.
[16] Baker, T., & Cavallo, P. (1999). Dynamic adaptation for deforming tetrahedral meshes. In 14th Computational Fluid Dynamics Conference (p. 3253).
[17] Dey, T. K., Edelsbrunner, H., Guha, S., & Nekhayev, D. V. (1998). Topology preserving edge contraction. In Publ. Inst. Math.(Beograd)(NS.
[18] Menon, S. (2011). A numerical study of droplet formation and behavior using interface tracking methods.
[19] Carling, J., Williams, T. L., & Bowtell, G. (1998). Self-propelled anguilliform swimming: simultaneous solution of the two-dimensional Navier-Stokes equations and Newton's laws of motion. Journal of experimental biology, 201(23), 3143-3166.