Computer-Aided Design 148 (2022) 103242

Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

A Parallel Variational Mesh Quality Improvement Method for Tetrahedral Meshes Based on the MMPDE Method☆,☆☆

Maurin Lopez [a], Suzanne M. Shontz [b,*], Weizhang Huang [c]

[a] Goodyear Innovation Center, The Goodyear Tire and Rubber Company, Colmar-Berg, Luxembourg
[b] Department of Electrical Engineering and Computer Science, Bioengineering Program, Information and Telecommunication Technology Center, University of Kansas, Lawrence, KS, USA
[c] Department of Mathematics, University of Kansas, Lawrence, KS, USA

ARTICLE INFO

Article history:
Received 31 January 2021
Received in revised form 11 December 2021
Accepted 25 February 2022

Keywords:
Parallel mesh quality improvement
Variational method
Tetrahedral mesh
Distributed computing

ABSTRACT

There are numerous large-scale applications requiring mesh adaptivity, e.g., cardiac electrophysiology, computational fluid dynamics, fracture propagation, and weather prediction. Parallel processing is needed for simulations involving large-scale adaptive meshes. In this paper, we propose a parallel variational mesh quality improvement algorithm for use with distributed memory machines. Our parallel method is based on the sequential method by Huang, Ren, and Russell and the recent implementation by Huang and Kamenski. Their approach is based on the use of the Moving Mesh PDE method to adapt the mesh based on the minimization of an energy functional for mesh equidistribution and alignment. This leads to a system of ordinary differential equations (ODEs) to be solved which determine where to move the interior mesh nodes. The MMPDE method successfully removes/reduces the number of extreme dihedral angles, particularly those less than 20^o or greater than 150^o. An efficient solution is obtained by solving the ODEs on subregions of the mesh with overlapped communication and computation. Strong and weak scaling experiments on up to 128 cores for meshes with up to 160M elements demonstrate excellent results.

1. Introduction

There are numerous large-scale scientific applications requiring adaptive meshes with millions to billions of elements, e.g., [1–6]. Such large computational simulations are possible due to the availability of massively parallel supercomputers which integrate central processing units (CPUs) and accelerators, such as graphics processing units (GPUs), Phi co-processors, and field programmable gate arrays (FPGAs). New parallel mesh generation, parallel mesh adaptation, and parallel mesh quality improvement algorithms have been developed to take advantage of these novel architectures.

Although there are numerous parallel mesh generation algorithms in existence (e.g., [7] and references therein and [8–12]), fewer parallel mesh quality improvement algorithms have been developed [13–20]. The methods presented in [13–15,18–20] are solely devoted to improving the mesh quality, whereas the ones in [16,17] combine mesh untangling and mesh quality improvement procedures. Our method is unique in that is a parallel variational approach for mesh quality improvement. It is also a serial–parallel consistent algorithm in that the same mesh quality results will be obtained independent of the number of cores.

Many parallel mesh adaptation methods have been proposed in recent years [21–31]; these methods construct a new mesh in parallel either by performing local modifications of an existing mesh or by analyzing the discretization error (through *a posteriori* error estimation) and using it to guide the remeshing. In either case, a metric is used to specify the stretching directions whenever anisotropic mesh adaptation is desired. The paper by Digonnet et al. is a new variational approach for mesh adaptation in parallel based on the use of an edge-based error estimator [30].

We also review the sequential methods developed for mesh quality improvement and mesh adaptation given the role in laying the foundations in these areas. The vast majority of such methods employ optimization techniques to improve the mesh quality or to adapt the mesh to changes in the geometry or the physics of the application. Optimization-based mesh quality improvement and mesh adaptation algorithms adjust the positions of the node while preserving the mesh topology [32–48] or alter the mesh topology while fixing the nodal positions [49,50].

Variational methods for mesh adaptation and mesh quality improvement have recently received considerable attention

☆ A preliminary version of a portion of these results appeared in shortened form in the Proceedings of the 28th International Meshing Roundtable.
☆☆ This paper has been recommended for acceptance by Joaquim Peiro, Suzanne Shontz & Ryan Viertel.
* Corresponding author.
E-mail addresses: maurin_lopez@goodyear.com (M. Lopez), shontz@ku.edu (S.M. Shontz), whuang@ku.edu (W. Huang).

https://doi.org/10.1016/j.cad.2022.103242
0010-4485/

from the meshing community (e.g., [46,47,51–56]). Whereas most optimization-based mesh quality improvement algorithms use gradient-based techniques to minimize an objective function, Huang and Kamenski [46,47] instead use the Moving Mesh PDE (MMPDE) method to discretize and find the minimum of an appropriately constructed meshing functional [57–59]. The minimizer of the meshing functional is a bijective mapping which generates an improved quality mesh as an image of the initial mesh.

In this paper, we present a novel, efficient parallel variational mesh quality improvement algorithm and the corresponding implementation for distributed memory machines [20]. Our parallel method is based on the sequential method by Huang, Ren, and Russell [51] and the recent implementation by Huang and Kamenski [47]. The method finds the minimizer of a meshing functional by solving a system of ordinary differential equations (ODEs) for the nodal velocities. We first review the key concepts of variational mesh methods and the implementation of the sequential MMPDE method in Section 2. In Section 3, we describe our parallel variational mesh quality improvement method for distributed memory systems, along with the implementation. Our method employs a domain decomposition approach in order to divide the workload among the cores. We reorganize the computation within each subregion in order to facilitate the overlap of communication with computation. We analyze the computational complexity of the method in Section 4. We carry out numerical experiments on tetrahedral meshes and determine the strong and weak scaling properties of the proposed method. The numerical experiments and the associated results pertaining to the mesh quality, runtimes (wall-clock time), and algorithmic scalability are discussed in Section 5. We also include results from an industrial example from the tire industry. We present our conclusions on the work and several potential directions for future work in Section 6.

2. Variational mesh adaptation methods

In this section, we present an overview of variational mesh adaptation and the corresponding methods. In the variational approach, an adaptive mesh is generated as the image of a reference mesh under a coordinate transformation. The coordinate transformation is determined as the minimizer of a meshing functional. The mesh concentration is typically controlled through a scalar or a matrix-valued function. This is referred to as the metric tensor or monitor function. Monitor functions are defined based on error estimates and/or physical considerations.

Several authors have reported on variational mesh adaptation methods with various types of meshing functionals. For example, Winslow [60] developed an equipotential method based on variable diffusion. Brackbill and Saltzman developed a method combining mesh concentration, smoothness, and orthogonality [61]. Dvinsky developed another approach based on the energy of harmonic mappings [62]. Variational methods based on the conditioning of the Jacobian matrix of the coordinate transformation were developed by Knupp [32] and Knupp and Robidoux [58]. More recently, equidistribution and alignment conditions were used by Huang [63] and Huang and Russell [64] to develop mesh adaptation methods.

The Moving Mesh PDE (i.e., MMPDE) method, which was proposed by Huang, Ren, and Russell in 1994 [51] is the basis upon which many other variational mesh adaptation methods have been developed. In 2015, Huang and Kamenski developed a more efficient implementation of the serial MMPDE method [47].

2.1. New implementation of the variational mesh adaptation method

In this subsection, we focus on Huang and Kamenski's new implementation of the MMPDE method [47]. Consider a domain $\Omega \subset \mathbb{R}^d$ ($d \geq 1$) and let $\mathcal{T}_h = \{K\}$ be a simplicial mesh containing N elements and N_v vertices on Ω. Denote the affine mapping $F_K : \hat{K} \to K$ and its Jacobian matrix by F_K', where $\hat{K}$ is the master element. Let the edge matrices for K and $\hat{K}$ be E_K and $\hat{E}$, i.e.,

$$E_K = [\mathbf{x}_1^K - \mathbf{x}_0^K, \ldots, \mathbf{x}_d^K - \mathbf{x}_0^K], \quad \hat{E} = [\xi_1 - \xi_0, \ldots, \xi_d - \xi_0],$$

where $\mathbf{x}_i^K$, $i = 0, \ldots, d$ and ξ_i, $i = 0, \ldots, d$ denote the coordinates of the vertices of K and $\hat{K}$, respectively. Notice that F_K', E_K, and $\hat{E}$ are related by $F_K' = E_K \hat{E}^{-1}$. Assume that a metric tensor (or a monitor function) $\mathbb{M} = \mathbb{M}(\mathbf{x})$ is given on Ω which provides directional and magnitude information for the elements.

A key idea of the MMPDE method is to view an adaptive mesh as a uniform one in the metric $\mathbb{M}$ in the sense that the size of all elements K in the metric $\mathbb{M}_K$ is the same, and all elements K in the metric $\mathbb{M}_K$ are similar to $\hat{K}$.

These two properties give rise to the equidistribution and alignment conditions:

$$|K|\sqrt{\det(\mathbb{M}_K)} = \frac{\sigma_h}{N}, \quad \forall K \in \mathcal{T}_h$$

$$\frac{1}{d}\mathrm{tr}\left((F_K')^T \mathbb{M}_K F_K'\right) = \det\left((F_K')^T \mathbb{M}_K F_K'\right)^{\frac{1}{d}}, \quad \forall K \in \mathcal{T}_h,$$

where $|K|$ is the volume of K, $\sigma_h = \sum_K |K|\sqrt{\det(\mathbb{M}_K)}$, and $\mathrm{tr}(\cdot)$ and $\det(\cdot)$ denote the trace and determinant of a matrix, respectively. Notice that $|K|\sqrt{\det(\mathbb{M}_K)}$ is the volume of K in the metric $\mathbb{M}_K$. Moreover, the first condition, the equidistribution condition, determines the size of elements through the determinant of $\mathbb{M}$. The larger $\det(\mathbb{M}_K)$ is, the smaller $|K|$ is. On the other hand, the second condition, the alignment condition, determines the shape and orientation of K through $\mathbb{M}_K$. Indeed, it can be shown [64] from the condition that the principal axes of the circumscribed ellipsoid of K coincide with the eigendirections of $\mathbb{M}_K$, and their semi-lengths are inversely proportional to the square root of the corresponding eigenvalues of $\mathbb{M}_K$.

Then an energy function for the equidistribution and alignment conditions is given by

$$I[\mathcal{T}_h] = \sum_K |K| \frac{1}{3}\sqrt{\det(\mathbb{M}_K)}\left(\mathrm{tr}(\mathbb{J}\mathbb{M}_K^{-1}\mathbb{J}^T)\right)^d + \sum_K |K| \frac{1}{3} d^d \sqrt{\det(\mathbb{M}_K)}\left(\frac{\det(\mathbb{J})}{\sqrt{\det(\mathbb{M}_K)}}\right)^2, \tag{1}$$

where $\mathbb{J} = (F_K')^{-1} = \hat{E}E_K^{-1}$. Minimization of the energy function will result in a mesh that closely satisfies the equidistribution and alignment conditions. It is worth pointing out [65] that $\mathrm{tr}(\mathbb{J}\mathbb{M}_K^{-1}\mathbb{J}^T)$ is mathematically equivalent to $1/a_{K,\mathbb{M}}^2$, where $a_{K,\mathbb{M}}$ denotes the minimum height of K in the metric $\mathbb{M}_K$. The appearance of this factor in the energy function plays a crucial role in preventing the mesh from tangling.

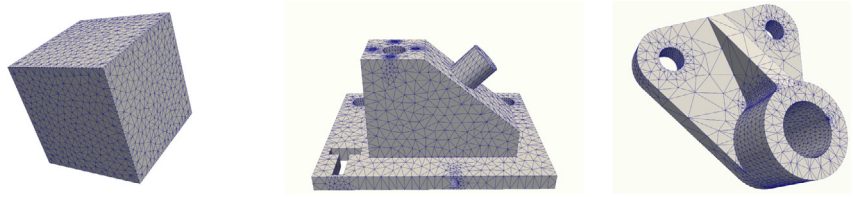
The MMPDE moving mesh equation is then defined as the (modified) gradient system of $I[\mathcal{T}_h]$, i.e.,

$$\frac{d\mathbf{x}_i}{dt} = -\frac{\det(\mathbb{M}_i)^{\frac{1}{d}}}{\tau}\frac{\partial I[\mathcal{T}_h]}{\partial \mathbf{x}_i}, \quad i = 1, \ldots, N_v,$$

where $\tau > 0$ is a parameter for adjusting the response time scale of mesh movement to the change in $\mathbb{M}$.

For mesh quality improvement, we choose $\mathbb{M} = \mathbb{I}$ (and $\tau = 1$), which means we want the mesh to be as uniform as possible in the Euclidean space. In this case, the moving mesh equation reads as

$$\frac{d\mathbf{x}_i}{dt} = \sum_{K \in \omega_i} |K| \mathbf{v}_{i_K}^K, \quad i = 1, \ldots, N_v, \tag{2}$$



(a) (b) (c)

Fig. 1. Example tetrahedral meshes for a cubic domain and two mechanical parts. The meshes contain 2636 nodes and 10,999 elements for (a), 5715 nodes and 22,155 elements for (b), and 3257 nodes and 11,464 elements for (c).

where ω_i is the element patch associated with $\mathbf{x}_i$, i_K is the local index for $\mathbf{x}_i$ on K, and $\mathbf{v}_{i_K}$ is the local nodal velocity contributed by K to the node $\mathbf{x}_i$. The analytical formula of the local nodal velocities is given in [47]. In the case when $\mathbb{M} = \mathbb{I}$,

$$\begin{aligned}\begin{bmatrix}(\mathbf{v}_1^K)^T\\ \vdots\\ (\mathbf{v}_d^K)^T\end{bmatrix} &= -G_K E_K^{-1} + E_K^{-1}\frac{\partial G_K}{\partial \mathbb{J}}\hat{E}E_K^{-1}\\ &\quad + \frac{\partial G_K}{\partial \det(\mathbb{J})}\frac{\det(\hat{E})}{\det(E_K)}E_K^{-1},\\ (\mathbf{v}_0^K)^T &= -\sum_{j=1}^{d}(\mathbf{v}_j^K)^T,\end{aligned} \tag{3}$$

where

$$\begin{aligned}G_K &= \frac{1}{3}\left(\mathrm{tr}(\mathbb{J}\mathbb{J}^T)\right)^d + \frac{1}{3}d^d\det(\mathbb{J})^2,\\ \frac{\partial G_K}{\partial \mathbb{J}} &= \frac{2d}{3}\left(\mathrm{tr}(\mathbb{J}\mathbb{J}^T)\right)^{d-1}\mathbb{J}^T,\\ \frac{\partial G_K}{\partial \det(\mathbb{J})} &= \frac{2}{3}d^d\det(\mathbb{J}).\end{aligned}$$

The nodal velocities of the boundary nodes are set to 0. They can also be modified to let the boundary nodes slide along the boundary. In our computation, the nodes at corners and on edges are fixed, while those on domain faces are allowed to slide along the faces.

To determine the locations of the interior nodes, Eq. (2) is then solved using the adaptive fourth-order Runge–Kutta–Fehlberg ODE solver (RKF45); see Section 3. It has been shown analytically and numerically in [65] that the mesh generated by the MMPDE moving mesh equation will stay nonsingular (i.e., no crossing nor tangling will occur) if it is nonsingular initially. In particular, for the energy function (1) with $\mathbb{M} = \mathbb{I}$, it is shown that the minimum height of elements is bounded below by

$$a_K \geq C_1 N^{-\frac{2}{d}}, \quad \forall K \in \mathcal{T}_h,$$

where C_1 is a positive constant depending on the value of the energy function at the initial mesh.

To perform other types of mesh adaptation, one simply needs to change $\mathbb{M}$ to something other than $\mathbb{I}$. Although this capability is part of the serial implementation, the parallel implementation is currently only capable of performing mesh smoothing.

2.2. Mesh examples for quality improvement

Numerical examples for mesh quality improvement using the MMPDE method described in the previous subsection have been reported in [66,67]. For completeness, we present several examples in three dimensions here. Fig. 1 shows three example meshes generated by TetGen [68] for a cubic domain and two mechanical parts, respectively. The MMPDE method is used (with the final time taken as $t = 1$) to improve the quality of these meshes. The distributions of the dihedral angles for the meshes before and after the smoothing are listed in Table 1. From the results, we can see that the MMPDE method is effective in removing/reducing the number of extreme angles and, in particular, those less than 20° or larger than 150°. This is consistent with observations made for other variational smoothing methods; e.g., see [34,39]. It is worth pointing out that this effectiveness depends on the geometry of the domain. For example, the MMPDE method eliminates nearly all angles less than 20° or larger than 150° for the cubic domain. On the other hand, the method eliminates nearly all of those angles less than 10° or larger than 160° for more complicated domains in Fig. 1(b) and (c). Interestingly, there are two angles larger than 170° in Fig. 1(c) that cannot be removed by the smoothing method.

Table 1
Distributions of dihedral angles for meshes before and after MMPDE smoothing.

For mesh in Fig. 1(a)					
Angle (°)	Before	After	Angle (°)	Before	After
0–10	**43**	**0**	90–100	5,357	5,117
10–20	**937**	**29**	100–110	3,460	3,677
20–30	2,246	1,464	110–120	2,211	2,686
30–40	4,862	6,341	120–130	1,369	2,035
40–50	8,277	9,508	130–140	921	1,150
50–60	10,480	10,640	140–150	567	128
60–70	9,974	9,295	150–160	**346**	**1**
70–80	8,349	7,750	160–170	**123**	**0**
80–90	6,472	6,173	170–180	0	0
For mesh in Fig. 1(b)					
Angle (°)	Before	After	Angle (°)	Before	After
0–10	**302**	**18**	90–100	11,334	10,966
10–20	**3,994**	**2,837**	100–110	6,731	6,973
20–30	7,987	8,421	110–120	5,299	5,622
30–40	12,090	13,486	120–130	4,333	4,823
40–50	15,341	15,678	130–140	2,784	3,553
50–60	16,903	16,856	140–150	1,712	1,583
60–70	14,946	15,141	150–160	**923**	**364**
70–80	15,978	14,485	160–170	**337**	**4**
80–90	11,936	12,120	170–180	0	0
For mesh in Fig. 1(c)					
Angle (°)	Before	After	Angle (°)	Before	After
0–10	**162**	**0**	90–100	6,584	5,688
10–20	**1,944**	**720**	100–110	3,846	4,216
20–30	4,220	4,730	110–120	3,049	3,218
30–40	6,003	6,696	120–130	1,876	2,168
40–50	6,651	7,673	130–140	1,032	1,449
50–60	8,972	8,420	140–150	650	478
60–70	8,349	8,554	150–160	**451**	**26**
70–80	8,405	7,796	160–170	**139**	**0**
80–90	6,449	6,950	170–180	2	2

3. Parallel variational mesh quality improvement algorithm

In this section, we present our novel parallel algorithm and implementation for distributed memory systems based on the moving mesh method described in the previous section.

3.1. Sequential algorithm

For the sequential algorithm, the adaptive fourth-order Runge–Kutta Fehlberg ODE solver (RKF45) with fifth-order error estimator is employed to solve Eq. (2) (see [69] for details). The RKF45 method approximates the solution of an ODE system in the form

$$\frac{dy}{dt} = f(t, y) \tag{4}$$

using a non-constant, optimal step size dt in each iteration. The method determines the step size dt in each iteration by comparing a fourth-order approximation, y_{i+1}, and a fifth-order approximation, z_{i+1}, to the solution. These approximations are given by

$$y_{i+1} = y_i + \frac{25}{216}k_1 + \frac{1408}{2565}k_3 + \frac{2197}{4104}k_4 - \frac{1}{5}k_5, \tag{5}$$

and

$$z_{i+1} = y_i + \frac{16}{135}k_1 + \frac{6656}{12\,825}k_3 + \frac{28\,561}{56\,430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6, \tag{6}$$

respectively. Here

$$\begin{aligned}
k_1 &= dtf(t_i, y_i),\\
k_2 &= dtf\left(t_i + \frac{1}{4}dt, y_i + \frac{1}{4}k_1\right),\\
k_3 &= dtf\left(t_i + \frac{3}{8}dt, y_i + \frac{3}{32}k_1 + \frac{9}{32}k_2\right),\\
k_4 &= dtf\left(t_i + \frac{12}{13}dt, y_i + \frac{1932}{2197}k_1 - \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3\right),\\
k_5 &= dtf\left(t_i + dt, y_i + \frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4\right),\\
k_6 &= dtf\left(t_i + \frac{1}{2}dt, y_i - \frac{8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 + \frac{1859}{4104}k_4 - \frac{11}{40}k_5\right).
\end{aligned} \tag{7}$$

The error is given by the ∞-norm of the difference between the two solutions, i.e., $\text{err} = \|z_{i+1} - y_{i+1}\|_\infty$. If err is smaller than a given tolerance, tol, then the solution, y_{i+1}, is accepted. One can show that the optimal step size is given by $q * dt$, where

$$q = 0.84\left(\frac{\text{tol} * dt}{\text{err}}\right)^{\frac{1}{4}}. \tag{8}$$

In Algorithm 1, (i.e., the algorithm for the method proposed in [47]), the calculation of the nodal velocities is directly related to the calculation of the k_i, which is the most computationally-intensive step (i.e., step 5). To calculate the values, k_i, in the RKF45 method, the algorithm loops over all elements calculating partial nodal velocities for each node. This requirement is the basis for our distributed data approach in the parallel algorithm.

Algorithm 1 Sequential variational mesh quality improvement algorithm.

```
1: Input: nodal coordinates, topology, boundary nodes
2: Define: Initial dt, t_final, tol, t = 0, i = 0
3: while (t < t_final) do
4:     for each node in the mesh do
5:         Compute k_1 − k_6 from Eq. (7) and the ODE in Eq. (2)
6:     end for
7:     Compute y_{i+1}, z_{i+1} (Eqs. (5) and (6)) and error (err)
8:     Compute dt = q * dt where q is given by Eq. (8)
9:     if (err < tol) then
10:        Accept y_{i+1} as a solution
11:    else
12:        Compute dt = max(q * dt, 0.1 * dt);
13:    end if
14:    Compute t = t + dt
15: end while
16: Output: new nodal coordinates
```

3.2. Overview of the parallel algorithm

In this subsection, we highlight three important aspects of our parallel algorithm (Algorithm 2): the distribution of the workload, the communication strategy, and the termination criteria. Although there exist multiple strategies to distribute the work among cores, we employ a domain decomposition approach in which we divide the domain into p regions, where p is the number of cores (i.e., steps 4 and 5 in Algorithm 2). Each region is (ideally) composed of one connected component; see Fig. 2. Fig. 2 illustrates an example of a domain decomposed into four regions where no edges are cut at the boundary between regions. We use this approach because according to Eq. (3) the nodal velocity of a particular node $\mathbf{x}_m$, such as the one in Fig. 3, is calculated based on the edge matrices of elements E_1, E_2, E_3, E_4, and E_5. Therefore, a decomposition of the elements of the domain into regions is the strategy that yields the best performance. To accomplish this, we use METIS [70], which is a library for partitioning meshes and graphs. We employed the mpmetis scheme to partition the mesh into regions so that each region has roughly the same number of elements and the number of interfaces between adjacent regions is minimized.

Once we have the mesh partition, core P_0 reads and distributes the information concerning the topology and nodal coordinates to the rest of the cores. In this step, each core creates a list (`SharedNodes_p[]`) whose size is equal to the number of nodes along partition boundaries (i.e., the number of shared nodes). Each core stores partial nodal velocities to specific locations in the list and fills-in the rest with zeros.

Whereas each core computes the new nodal positions for the interior nodes of its corresponding region, the new nodal positions for nodes along partition boundaries (corresponding to the shared nodes) require communication and verification steps (i.e., steps 4 and 6 in Algorithm 3). In our parallel algorithm, all communication steps are reduction operations. To compute the new nodal positions for shared nodes, we perform a reduction operation (summation) over the list `SharedNodes_p[]` in which we store the partial nodal velocities of the shared nodes. Fig. 4 illustrates this process; it shows a domain which is partitioned into four regions. For simplicity, we assume that v_1, v_2, v_3, v_4 and v_5 are the only nodes shared among the regions. A reduction operation (summation) over the list `SharedNodes_p[]` will provide to every processor the full nodal velocity for the five nodes in the boundary. We also need communication steps to calculate the global error (i.e., step 16 in Algorithm 2) and the average mesh

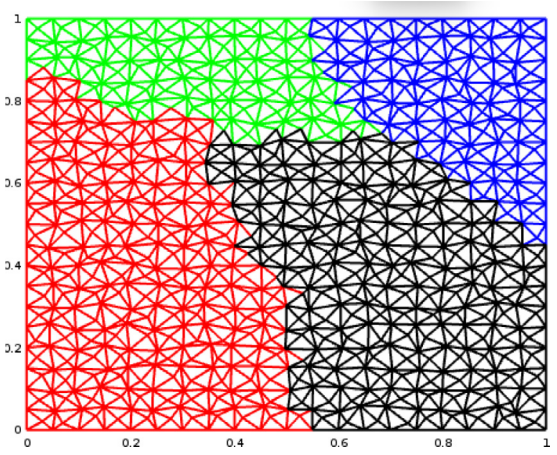


Fig. 2. Example of a 2D domain partitioned into four regions. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

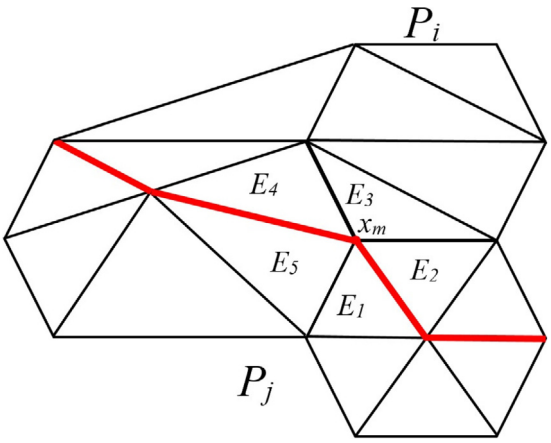


Fig. 3. Patch of elements with $\mathbf{x}_m$ as one of its vertices.

quality (i.e., step 24 in Algorithm 2). To calculate the global error, we require a reduction operation to calculate the maximum. We also require a summation reduction for the average mesh quality.

Finally, we employ a tetrahedral mesh quality metric in order to evaluate the quality of the mesh on each iteration. We utilize the mesh quality information in the termination criteria. The mesh quality metric (a version of the aspect ratio) implemented in our algorithm is given by

$$q = \frac{CR}{3 * IR}, \tag{9}$$

where CR is the circumsphere radius, and IR is the inscribed sphere radius. For this metric, $q \in [1, \infty)$ where $q = 1.0$ is the optimal mesh quality. We terminate the parallel variational mesh quality improvement algorithm when the difference in the average mesh quality on two consecutive iterations is small (i.e., less than a specified tolerance).

3.3. Overlapping communication with computation

As we mentioned before, the most computationally-intensive step in the parallel variational mesh quality improvement algorithm is the computation of the nodal velocities. For the mesh in Fig. 3, core P_i is unable to compute the nodal velocity for node $\mathbf{x}_m$ because the core does not have access to elements E_1 and E_5.

Algorithm 2 Parallel algorithm for variational mesh quality improvement.

1: Input: nodal coordinates, topology, boundary nodes, domain decomposition information
2: Define: Initial dt, $errtol$, tol, and $t = 0$
3: // Mesh partition, and data structure creation
4: Partition the mesh using METIS
5: Create and distribute data structures among cores
6: Compute: local and global mesh quality Q_{new} using **MPI_Iallreduce**
7: Split elements into two sets, `Elements_interior[]` and `Elements_bndry[]`
8: Check that the communication is completed (**MPI_Wait**)
9: Set $Q_{old} = 1.0$
10: // Solve differential equation Eq. (2)
11: **for** all p cores in parallel **do**
12: **while** $(|Q_{old} - Q_{new}| > errtol)$ **do**
13: $Q_{old} = Q_{new}$
14: Compute k_i using Algorithm 3
15: Compute y_{i+1}, z_{i+1} (Eq. (5) and Eq. (6))
16: Compute local error (err) and apply **MPI_Allreduce** to obtain global error
17: Compute $dt = q * dt$, where q is given by Eq. (8)
18: **if** (err < tol) **then**
19: Accept y_{i+1} as a solution
20: **else**
21: Compute $dt = \max(q * dt, 0.1 * dt)$
22: **end if**
23: Compute $t = t + dt$
24: Compute: local and global mesh quality Q_{new} using **MPI_Iallreduce**
25: **end while**
26: **end for**
27: Output: New nodal coordinates

Algorithm 3 Subroutine to compute k_i in Algorithm 2.

1: **for** i=1 to 6 **do**
2: Compute k_i from Eq. (7) using only nodes from `Elements_bndry[]`
3: Copy shared nodes (from k_i) to a global shared node array in p
4: Communicate and sum all global shared node arrays (**MPI_Iallreduce**)
5: Compute k_i from Eq. (7) using only nodes from `Elements_interior[]`
6: Check that the communication has been completed (**MPI_Wait**)
7: Update k_i with new shared node information
8: **end for**

Therefore, P_i calculates only a portion of the nodal velocity at this node. The same is true for core P_j.

According to the previous description, we design the parallel algorithm such that every core P_i loops once over its own elements to calculate the nodal velocities for the interior nodes within a region. However, for shared nodes, the nodal velocities are incomplete. Therefore, in this case, e.g., for $\mathbf{x}_m$ in Fig. 3, cores P_i and P_j store the partial nodal velocities in the `SharedNodes_p[]` list. Finally, the nodal velocities for the shared nodes require a reduction operation (summation) over `SharedNodes_p[]` and a verification step (i.e., steps 4 and 6 in Algorithm 3).

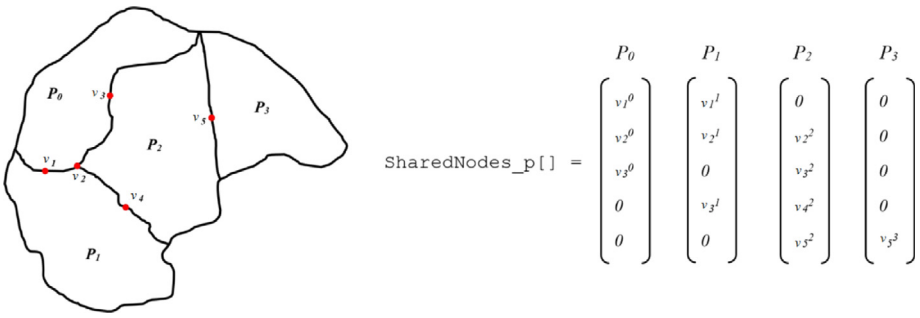


Fig. 4. Example of a 2D domain with a partition composed of four regions which share only the nodes in red, i.e., v_1, v_2, v_3, v_4 and v_5.

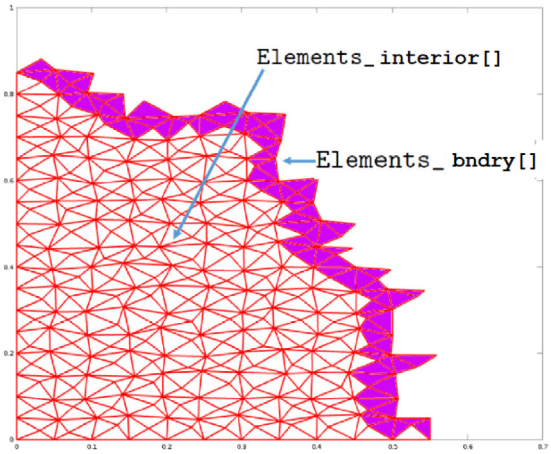


Fig. 5. The lower-left region of the domain in Fig. 2 is further subdivided into two data structures to allow the overlap of communication with computations.
.

It is possible to overlap the communication and computation and reduce the overall runtime by reorganizing the data in the data structures and the employment of non-blocking communication MPI commands. In a non-blocking communication strategy, the algorithm does not wait for a communication to be completed once it is initiated. Thus, instead of waiting to complete a send-receive structure, the algorithm will continue working and it will check regularly if the communication was completed. To this end, each core splits the list of local elements `Elements_proc[]` into two new lists, i.e., `Elements_interior[]` and `Elements_bndry[]` (i.e., step 7 in Algorithm 2). The algorithm stores the elements that contain at least one shared node in the data structure `Elements_bndry[]`, whereas the elements whose nodes are interior nodes are stored in `Elements_interior[]`. To illustrate this internal subdivision within each region, let us consider the case of the 2D domain divided into four regions depicted in Fig. 2. The region in red (i.e., the lower-left region) is further subdivided as shown in Fig. 5. Thus, we calculate the nodal velocities in two steps (i.e., steps 2 and 5 in Algorithm 3). After the first step, the algorithm will have partial nodal velocities for the shared nodes. Therefore, we can initiate the communication using the non-blocking collective command **MPI_Iallreduce** and simultaneously we calculate the nodal velocities for the interior nodes (i.e., steps 4 and 5 in Algorithm 3). Once the algorithm finishes the calculation of nodal velocities for the interior nodes, the algorithm checks to see if the communication has completed using **MPI_Wait** (i.e., step 6 in Algorithm 3). Finally, the algorithm updates the nodal velocities for the shared nodes (i.e., step 7 in Algorithm 3).

4. Parallel runtime analysis

In this section, we discuss the runtime performance of the parallel algorithm described in the previous section. In particular, we analyze the average parallel runtime.

First, we assume the partition of the initial mesh is given to the algorithm as input data. Recall that we use METIS to accomplish this step. Once the algorithm reads the input data, core P_0 distributes the information among cores according to the partition file. This overhead computation is performed sequentially and occurs just once throughout the execution of the algorithm. We assume this step takes t_N time.

Since we performed the mesh partitioning step over the elements of the mesh, assuming that N is the total number of mesh elements, each core contains (ideally) $\lceil N/p \rceil$ elements. With this information, the splitting operation performed within each region to overlap communication with computation takes $\lceil N/p \rceil (d+1)$ operations, where d is the dimension. This step is also performed once in the algorithm.

For the next step, we solve the differential Eq. (2) by calculating the values k_i. To calculate k_i, the algorithm loops over the mesh elements. If the maximum time to calculate the nodal velocity for each node is t_{vn}, then the total serial time to calculate the nodal velocities is $N(d+1)\,t_{vn}$. Therefore, the parallel time is $\lceil N/p \rceil (d+1)\,t_{vn}$. Moreover, we define the number of elements containing at least one shared node in the region corresponding to core P_i as $N_{sh}^{(P_i)}$, and the number of elements containing only interior nodes as $N_{int}^{(P_i)}$. Note that $\lceil N/p \rceil = N_{sh}^{(P_i)} + N_{int}^{(P_i)}$. For the communication process, first, we extract the local shared nodes. Assuming the time to copy one node from the local to the global list is t_c and the number of shared nodes in the mesh is V_{sh}, then this step takes $V_{sh}\,t_c$ time. Similarly, assuming that the time to send a vector with V_{sh} nodes is t_s, the communication process takes $\log_2(p)\,t_s + p\,V_{sh}\,t_c$. Note that this was implemented as a non-blocking communication process using the computation time $N_{int}^{(p)}(d+1)\,t_{vn}$ to overlap communication and computation. Thus, the time to compute these two processes is Tc_{total}, where

$$Tc_{total} = \begin{cases} T_{int}, & \text{if } T_{int} > T_{comm}, \\ T_{int} + |T_{int} - T_{comm}|, & \text{otherwise.} \end{cases} \tag{10}$$

Here $T_{int} = N_{int}^{(p)}(d+1)\,t_{vn}$ and $T_{comm} = \log_2(p)\,t_s + p\,V_{sh}\,t_c$.

To copy the information from the global to the local list in each core costs $V_{sh}\,t_c$. Assuming that the time to compute the error in each coordinate of each node is t_e, the total serial time is $N(d+1)d\,t_e$. In parallel, it is $\lceil N/p \rceil (d+1)d\,t_e$ plus the time to calculate the maximum error among cores, which is $\log_2(p)$. Finally, if t_q is the time to calculate the quality of one element, then Nt_q is the time to calculate the quality for the serial algorithm, and $\lceil N/p \rceil t_q$ is the time for the parallel one. The time to calculate the average quality among cores is $\log_2(p)$. With this information, the total

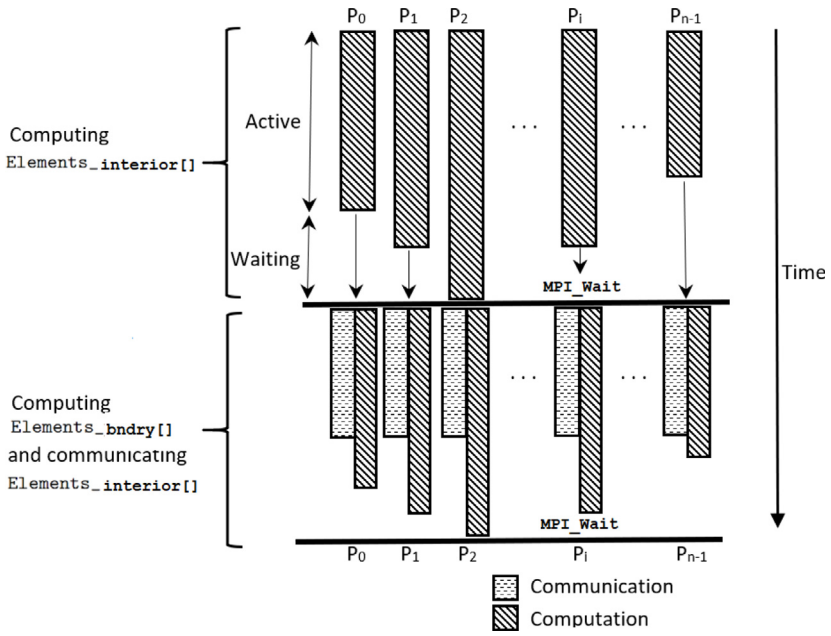


Fig. 6. Schematic representation of Algorithm 3.

parallel time per iteration is

$$T_P = p\,V_{sh}\,t_c + 2\log_2(p) + (d+1)(\lceil N/p\rceil_{sh}\,t_{vn} + \lceil N/p\rceil d\,t_e) + \lceil N/p\rceil t_q + Tc_{total}. \quad (11)$$

At this point, we should examine Algorithm 3 which is responsible for the majority of the computations (and possibly the major source of overhead due to communication) in Algorithm 2. A schematic representation of Algorithm 3 is shown in Fig. 6. Recall that the computation of k_i is divided in two steps. In the first one, k_i is computed on shared nodes stored in `Elements_bndry[]`. Although the size of the list `Elements_bndry[]` is approximately equal on each core, we still need to synchronize before starting the communication process and the computation of k_i in `Elements_interior[]` in the second step. In the latter, we expect to reduce the overhead due to communication by overlapping communication with computations. The major source of overhead due to communication in Eq. (11) is Tc_{total} and $p\,V_{sh}\,t_c$ due to the fact that the number of shared nodes always increases with the number of mesh elements and the number of cores; therefore, $p\,V_{sh}\,t_c$ increases. Hence, excellent timing results are expected in the cases for which the number of interior nodes in each partition is large compared with the number of shared nodes in the mesh.

5. Numerical experiments

Our algorithm was implemented in C/C++ using the message-passing interface (OpenMPI version 1.8.7). We ran our experiments on the high performance computing cluster available to us through the Advanced Computing Facility (ACF) at the University of Kansas. More specifically, we ran the experiments on twenty-one Dell R730 servers, each of them equipped with 2x dodeca-core Intel Haswell processors running at 2.5 GHz with 128 GB of RAM, 1TB HDD, and FDR Infiniband. It should be noted that we employed any subset of cores that were available to us for our experiments. For example, when running on 32 cores, our code may have been assigned by the scheduler to run on 20 cores on node 0 and 12 cores on node 1, on 16 cores on node 0 and 16 cores on node 1, on two cores each for nodes 0 to 15, or another such arrangement. This has implications for the communication time (and hence the runtime), as the time required for intra-node communication is less than that required for inter-node communication. Hence our experiments involve averaging of the runtimes from several runs of the code.

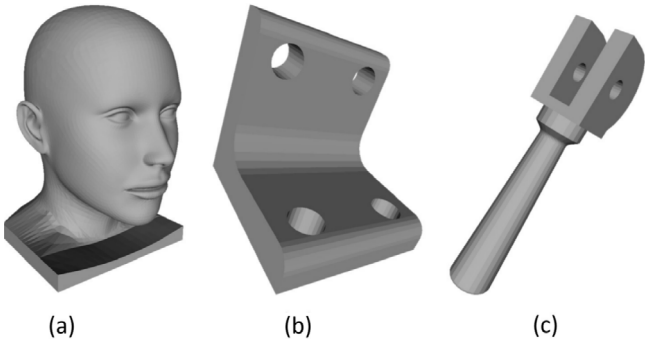


Fig. 7. Domains used to test the parallel algorithm: (a) bust, (b) bracket and (c) double cam tool.

To test the performance of our parallel algorithm, we constructed several tetrahedral meshes based on three geometries from various applications and with different characteristics. Fig. 7 illustrates the three-dimensional domains used in our experiments. We chose the geometries from different online databases: Fig. 7(a) is part of the 3dcadbrowser project [71], while Fig. 7(b) and (c) are part of the French Institute for Research in Computer Science and Automation (INRIA) databases [72]. We used GHS3D [72] and MeshLab [73] to generate a new surface mesh and to scale the domain to meet our needs. Based on these surface meshes, we generated tetrahedral volume meshes using Tetgen [68] with the numbers of elements specified in Tables 2 and 3. Finally, we randomly perturbed the nodes of each mesh to reduce their quality. The resulting tetrahedral meshes were then used to test the performance of our parallel variational mesh quality improvement (Parallel VMQI) algorithm. We used the meshes for the bust and the double cam tool domains to test the algorithm for strong scaling, whereas the meshes for the bracket domain were employed to test weak scaling.

Table 2
Size of tetrahedral meshes for the bust and the double cam tool domains.

Mesh	# Nodes	# Elements
Bust	12,895,493	80,000,012
Double cam tool	7,089,753	41,405,684

Table 3
Various mesh sizes for the bracket domain.

Mesh	# Nodes	# Elements
	450,960	2,500,032
	864,028	5,000,025
Bracket	1,716,222	9,999,990
	3,269,784	19,999,978
	6,497,224	40,000,000
	12,957,609	80,000,037
	24,177,335	159,745,245

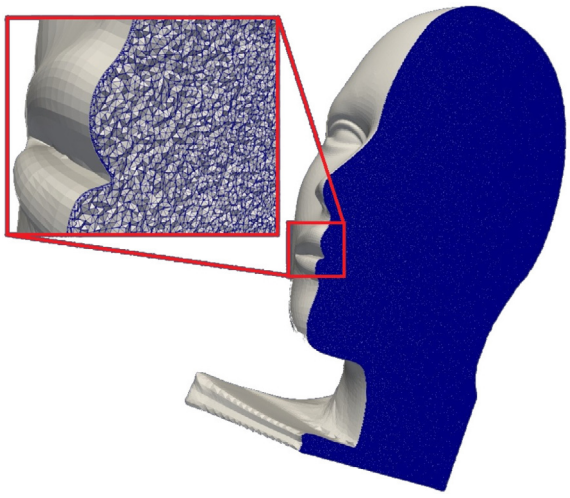


Fig. 8. 80M element tetrahedral mesh of the bust domain.

For our first experiment, we employed a tetrahedral mesh with approximately 80M elements for the bust domain (see Fig. 8). We ran the algorithm with various numbers of cores using $dt = 10^{-14}$, errtol $= 10^{-5}$, and tol $= 0.001$ as input parameters (see Algorithm 2). These values guarantee that the algorithm will run until convergence with an absolute error of less than errtol $= 10^{-5}$. Fig. 9 shows the average mesh quality versus the number of iterations. This demonstrates the ability of the algorithm to improve the average mesh quality as measured by the aspect ratio beta metric.

Fig. 10(a) and (b) show that for a small number of cores, the runtime, and speedup achieved by the parallel algorithm are very close to the ideal ones. A small deviation from the ideal speedup for a larger number of cores is also observed. The deviation is more pronounced at sixteen cores, and it does not grow much for a greater number of cores. It is clear that the pre-processing step (distribution of nodes, elements, and boundary nodes and identification of shared nodes) is a major source of overhead that significantly contributes to the discrepancy between the calculated and ideal speedup. On the other hand, when the number of interior nodes on each core is high compared with the number of shared nodes, it is more likely that the communication steps (when solving the differential equations) overlaps with the calculations of the nodal velocities for the interior nodes; therefore, the communication steps contribute less to the performance degradation in such a case. The runtimes reported in Fig. 10 are the average obtained from five runs; the numerical values are reported in Table 4. Note that the runtime decreased from

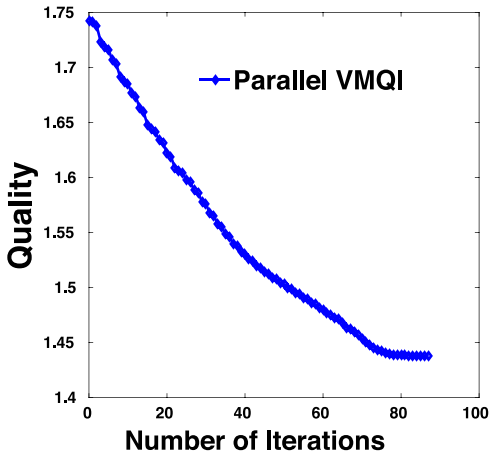


Fig. 9. Average quality versus number of iterations for the 80M element tetrahedral mesh of the bust domain.

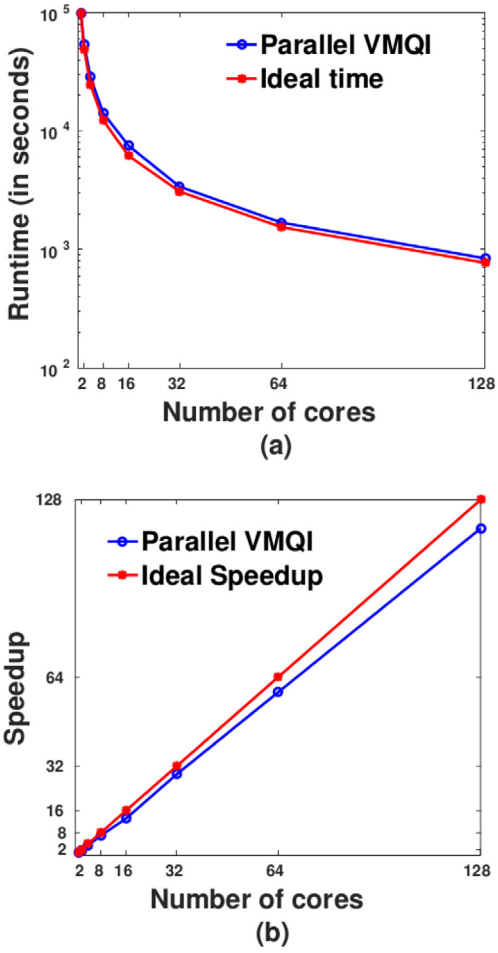


Fig. 10. (a) Total runtime and (b) speedup for the Parallel VMQI algorithm for the 80M element tetrahedral mesh of the bust domain.

nearly 28 h on 1 core to approximately 14 min on 128 cores. It is expected that the runtime would decrease further if more than 128 cores were employed.

Our second experiment is a strong scaling experiment using the double cam tool domain and a tetrahedral mesh with approximately 40M elements (see Fig. 11), which is approximately half the number of elements used for the first experiment. We

Table 4
Runtime values (average for five runs) for the bust geometry for various numbers of cores.

# Cores	Time (s)
1	100,645
2	54,257
4	28,554
8	14,179
16	7,507
32	3,390
64	1,685
128	839

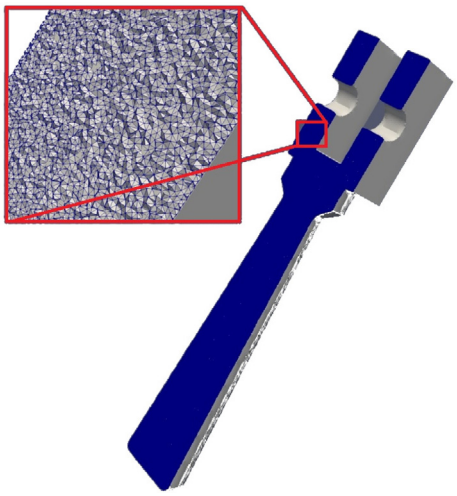


Fig. 11. 40M element tetrahedral mesh of the double cam tool domain.

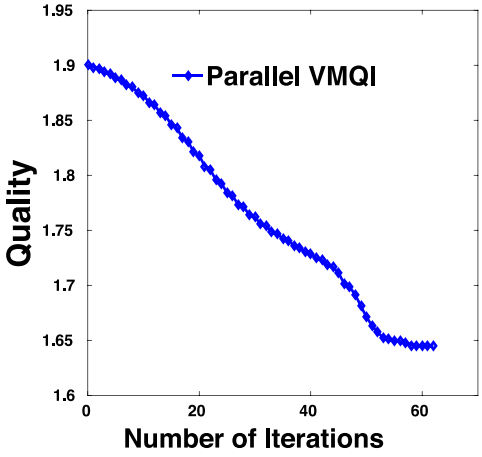


Fig. 12. Average quality versus the number of iterations for the 40M element tetrahedral mesh of the double cam tool domain.

decided to include this test case so as to measure the performance of the algorithm when the number of interior nodes per core is reduced. In this case, it may be more challenging to overlap communication with computation in an effective manner. The initial parameters (dt, errtol, tol) are the same as in the first strong scaling test. Fig. 12 shows the average mesh quality versus the number of iterations for this tetrahedral mesh.

Fig. 13(a) and (b) show the total runtime and speedup for the 40M element mesh of the double cam tool. The numerical runtime values are reported in Table 5. The results are, in general, similar to the ones for the first test case. For this mesh, the algorithm required approximately 10 h on one core; the runtime decreased to approximately 5 min on 128 cores.

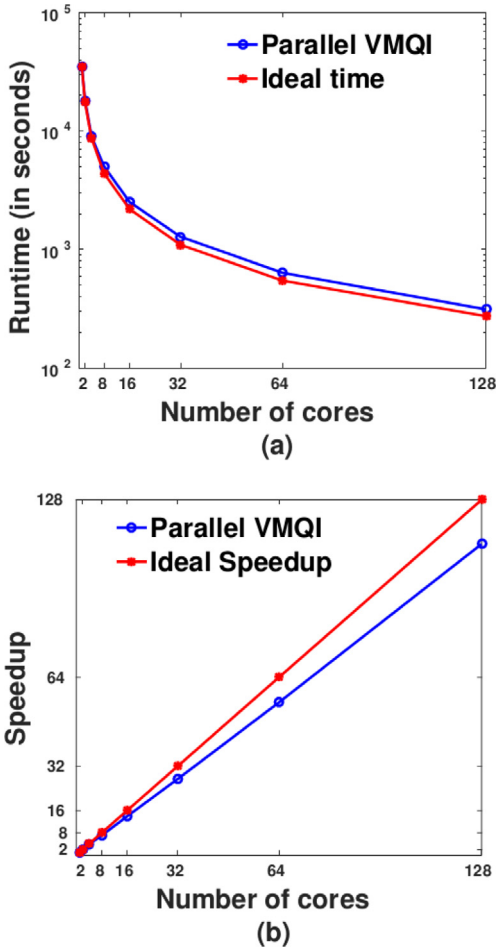


Fig. 13. (a) Total runtime and (b) speedup for the Parallel VMQI algorithm for the 40M element tetrahedral mesh of the double cam tool domain.

Table 5
Runtime values (average for five runs) for the double cam tool geometry for various numbers of cores.

# Cores	Time (s)
1	35,990
2	18,094
4	9,037
8	5,009
16	2,512
32	1,277
64	637
128	312

Again, further decrease in the runtime is expected if additional cores are employed. These results demonstrate that our parallel algorithm scales very well with the high-performance computing resources utilized at the University of Kansas. It is worth noting that the maximum number of cores reported in our experiments was limited by cluster size and accessibility at the time of our experiments.

We have also computed the distributions of dihedral angles for meshes in Figs. 8 and 11 before and after smoothing (see Table 6). Similar to the examples in Section 2, the algorithm is able to reduce significantly the number of extreme angles, in particular, those less than 10° or larger than 160°.

We attribute the good results from the previous two examples to the ability of our parallel algorithm to overlap communication with computation, thus reducing the runtime. When this is

Table 6
Distributions of dihedral angles for meshes in Figs. 8 and 11 before and after parallel MMPDE smoothing.

For mesh in Fig. 8					
Angle (°)	Before	After	Angle (°)	Before	After
0–10	**71,830**	**12,831**	90–100	102,330,333	104,250,999
10–20	**812,680**	**258,845**	100–110	16,986,746	14,925,345
20–30	7,658,455	7,857,632	110–120	10,205,043	10,583,342
30–40	17,793,713	17,576,532	120–130	7,236,446	7,167,455
40–50	33,470,968	32,721,591	130–140	4,840,265	4,817,223
50–60	62,018,559	61,957,342	140–150	3,036,390	3,084,461
60–70	80,980,238	82,530,926	150–160	1,138,502	984,549
70–80	73,574,498	73,544,091	160–170	**75,808**	**521**
80–90	57,761,175	57,726,091	170–180	**8,422**	**295**
For mesh in Fig. 11					
Angle (°)	Before	After	Angle (°)	Before	After
0–5	**36,836**	**6,580**	80–110	53,502,735	53,462,051
5–10	**416,759**	**132,741**	110–120	8,711,152	8,679,664
10–20	3,927,413	4,029,555	120–130	5,694,894	5,632,483
20–30	9,124,981	9,013,606	130–140	3,714,280	3,675,618
30–40	17,164,599	16,803,893	140–150	2,482,187	2,470,371
40–50	31,804,389	31,772,996	150–160	1,557,139	1,581,775
50–60	41,548,840	42,323,552	160–170	583,847	505,404
60–70	37,986,922	38,227,739	170–175	**38,876**	**85**
70–80	30,133,936	30,115,944	175–180	**4,319**	**47**

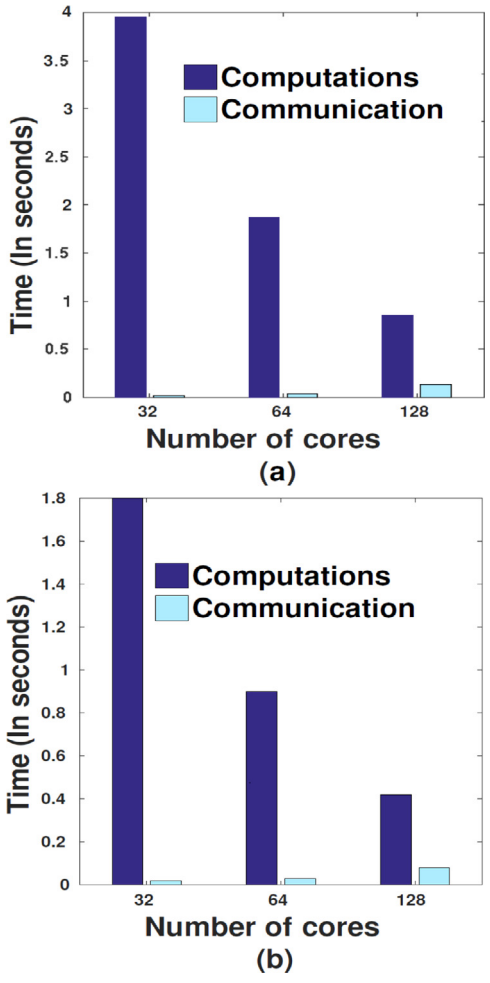


Fig. 14. Communication and computation times employed for one iteration to calculate the nodal velocities in one region of (a) 80M element mesh of the bust domain and a (b) 40M element mesh of the double cam tool domain.

possible, the major source of performance degradation, i.e., Tc_{total} from Eq. (10), is reduced. Fig. 14(a) and (b) show the computation

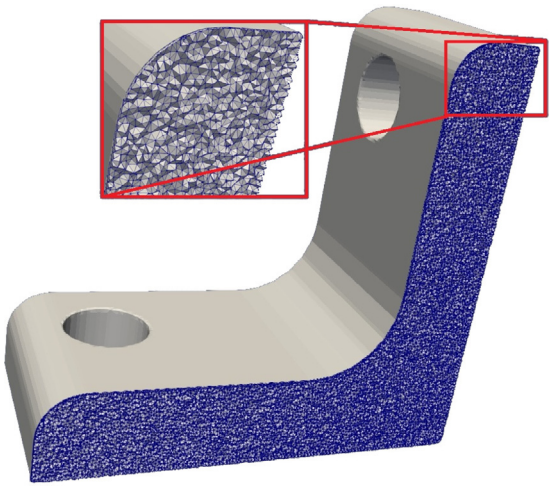


Fig. 15. 20M element tetrahedral mesh of the bracket domain.

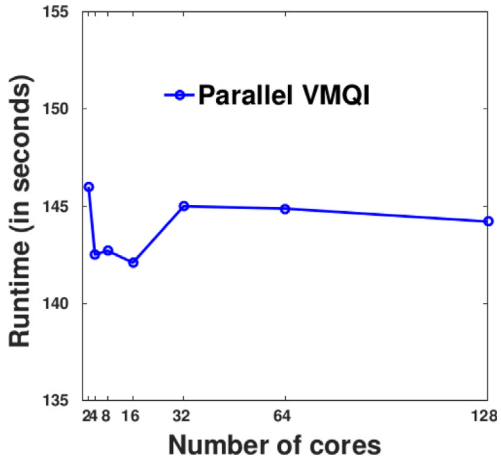


Fig. 16. Runtime versus number of cores to test the weak scaling efficiency.

and communication time for the bust and double cam tool test cases. The computation time is the time used by one core to calculate the nodal velocities for the interior nodes in its own region; the communication time is the time employed to communicate the shared nodes. Note that, for these two cases, the computation time is always significantly higher than the communication time, which guarantees good algorithmic performance. Also, it is expected that the computation time is reduced by half each time we double the number of cores. However, the communication time does not show a clear growth pattern. Theoretically, for the ideal case, the communication time should exhibit logarithmic growth, but in practice this is not the case. For our case, the communication time is related to the architecture of the cluster and with the distribution and availability of nodes and cores at runtime. More tests are needed with a greater number of cores and various mesh sizes to examine the growth in communication time to determine if complete overlap is still possible. It is important to note (based on the previous results) that it is very likely that good speedup results will be obtained when employing up to 128 cores for any mesh size. This number of cores is often sufficient for simulations involving practical engineering applications.

We also performed a weak scaling test, which investigates how the solution time changes with respect to an increasing number of cores (and assuming a constant workload per core), using various tetrahedral meshes for the bracket domain (see

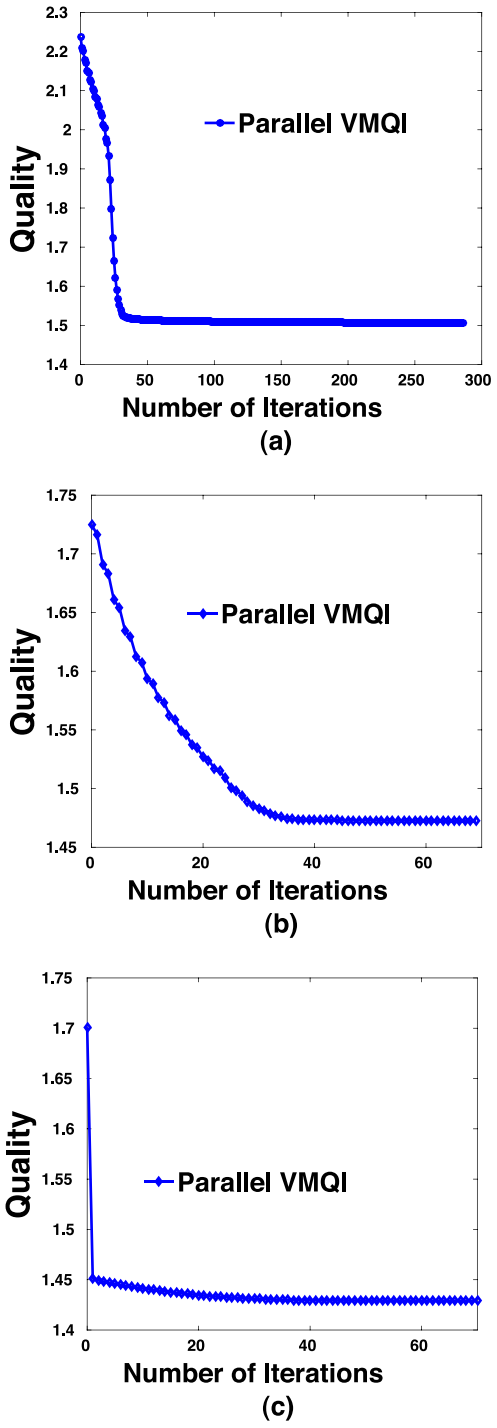


Fig. 17. Quality versus number of iterations for the bracket domain with (a) 2.5M, (b) 10M, and (c) 40M elements.

Table 3 and Fig. 15). For this experiment, we used the same parameters as in the previous test case, except for the initial dt value, which was $dt = 10^{-6}$ for this case. We made this change to better control the number of iterations in each computational simulation. Fig. 16 shows the weak scaling result for the algorithm. We observe a small oscillation in the runtime for low core counts. This oscillation, or deviation from the mean, is at most six seconds; this is a deviation of less than 5% from the mean value. This behavior is typical of a weak scaling result stemming from unstructured mesh computations, as it is very difficult to double exactly the problem size as the number of cores is doubled. Also, since the number of iterations for each simulation might be different (see Fig. 17), the results shown in Fig. 16 correspond to the time Algorithm 2 takes to run ten iterations. Numerical runtime values are reported in Table 7.

Fig. 18. Cured tire geometry from www.yeggi.com.

Table 7
Runtime values (for ten iterations) for the bracket geometry for the weak scaling test.

# Cores	Time (s)
2	146
4	143
8	143
16	142
32	145
64	145
128	144

5.1. Mesh smoothing for industrial simulations problems: Compression molding applications

The compression molding method is a manufacturing process in which a moldable material, usually heated, is placed into a cavity and then pressed to get into the shape of the cavity. Among the most common materials used in the compression molding process are thermoplastics and rubber type compounds.

Materials used in compression molding can be perceived as solid materials undergoing some type of deformation, but, they can be better viewed as very high viscous non-Newtonian fluids flowing due to some forced exerted on then. The fluid will flow according to the shape of the cavity or mold.

In this subsection, we consider an example of a step commonly seen in the tire manufacturing process. For this, we will use the tire geometry in Fig. 18, which is a cured tire, i.e., the end result of the tire manufacturing process.

In the tire manufacturing process, before we have the cured tire (Fig. 18), as we all use in our cars, we have what is called green tire. In the latter, the rubber compound is in a softer state, and the tire has no pattern (see Fig. 19(a)). The final pattern of the tire is given by the metallic mold part (see Fig. 19(b)) which is placed around the green tire as shown in Fig. 19(c). Finally, the green tire is inflated from the inside by inflating the bladder, thus pushing the soft rubber against the metallic mold. A 2D schematic of this process is shown in Fig. 20.

This process can sometimes be modeled using numerical simulations. By doing this, we can predict the pattern on the cured

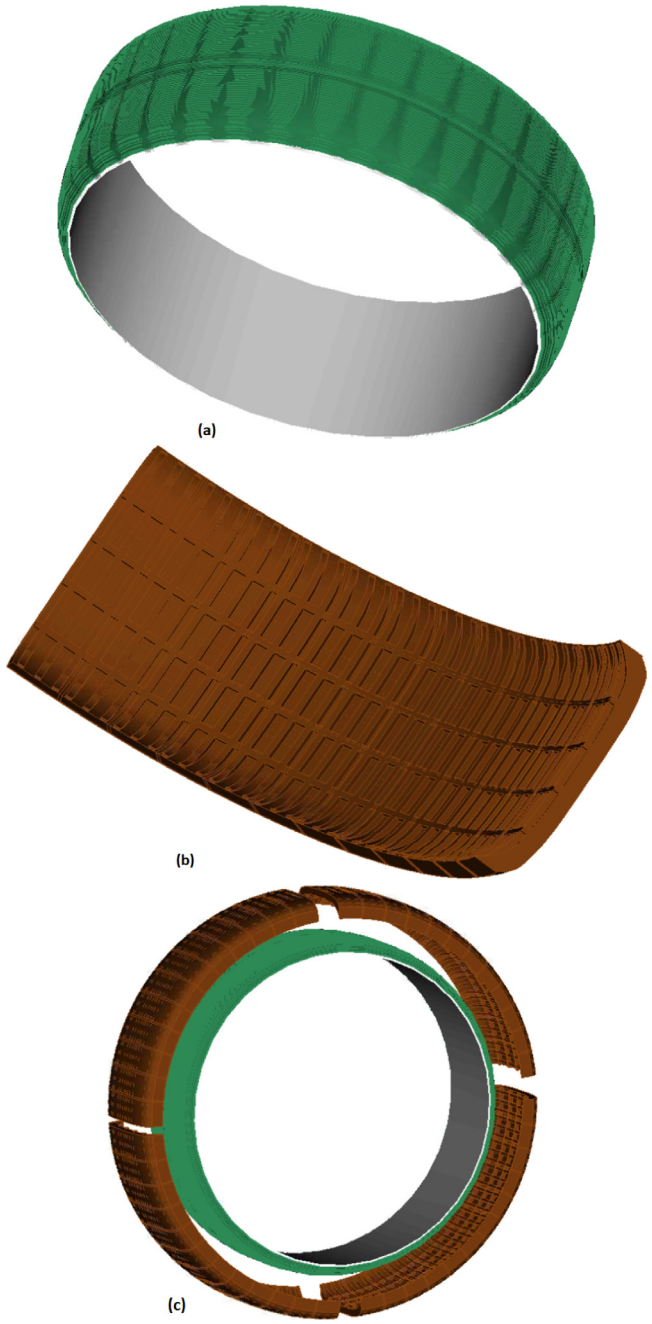


Fig. 19. Cure step in the tire manufacturing process (a) green tire, (b) mold with pattern, and (c) mold around the green tire. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

tire and spot possible anomalies before the tire is manufactured. One way to perform the simulation is to treat the green tire as a solid and the simulation as a contact problem. In this way, the green tire is meshed separate from the mold, and when the meshes overlap, the green tire mesh deforms to adjust to the mold mesh. This produces a deformation of the mesh, and the mesh quality decreases (see Fig. 21). In the majority of the cases, the quality of the mesh is improved (or the mesh is refined) in between simulation steps to prevent divergence of the simulation.

We have performed such a simulation and use one intermediate step where the quality of the mesh has decreased as the

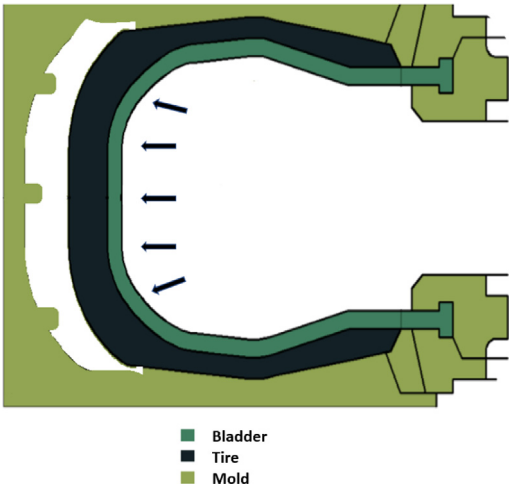


Fig. 20. Bladder inflates to push the green tire against the mold to create the final pattern in the tire. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

starting point to test our parallel algorithm trying to mimic the same conditions as is done in many companies.

For this experiment, we employed a tetrahedral mesh with approximately 500K elements. We ran the parallel algorithm on a workstation-server equipped with 4 Intel Xeon Silver 4210 processors running at 2.2 GHz with 64 GB of RAM and 2TB HDD. We tested the algorithm using up to 32 cores. We used input parameters similar to our previous experiments for consistency. The results in Fig. 22 demonstrate again the ability of our algorithm to improve the quality of the mesh. Also, it is worth mentioning that since the parallel algorithm is serial–parallel consistent, the quality of the resulting mesh is always the same independent of the number of cores used to run the parallel algorithm.

The speedup plot in Fig. 23 also confirms the good results of the parallel algorithm, as was already demonstrated by the previous examples. It is worth mentioning that the timing in Fig. 23(a) corresponds to the average of 5 runs on each core. Fig. 24 shows the wall clock time for 2 and 8 cores as an example. It is important to recall that the excellent results we obtained for the speedup are a consequence of the algorithm's ability to overlap communication and computation depending on the number of cores used. We have observed in this paper that overlapping them is possible when the algorithm is run on up to 128 cores, which is a greater number of cores than is typically used for industrial applications.

6. Conclusions and future research

We proposed a parallel variational mesh quality improvement algorithm and an associated implementation for the method in [47,51] for distributed memory machines. To distribute the workload among cores, we use METIS to partition the mesh into regions of connected elements. The algorithm identifies the elements in each region that contain at least one node that is shared by multiple regions (shared nodes). After distribution of the data (i.e., nodal coordinates, topology, and boundary nodes), each core organizes its corresponding elements into two sets, i.e., the elements composed of only interior nodes and the elements which have at least one shared node.

We employed the RKF45 method to solve the system of ODEs associated with the interior nodes. For this process, the parallel algorithm loops over all elements on each core to calculate

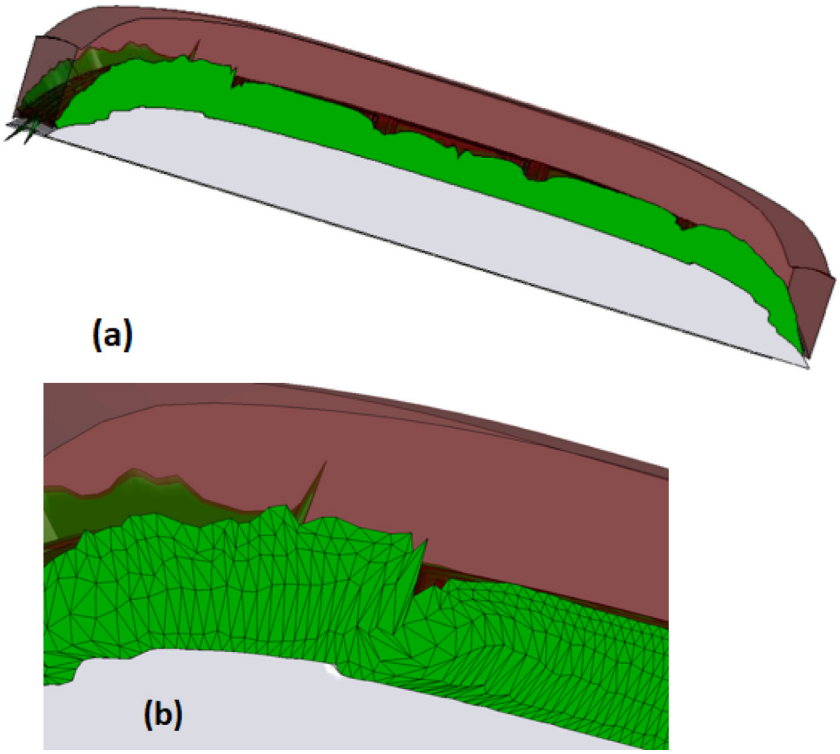


Fig. 21. Simulation results after the green tire is pushed against the mold and is deformed: (a) one section of the deformed tire; (b) deformed mesh. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

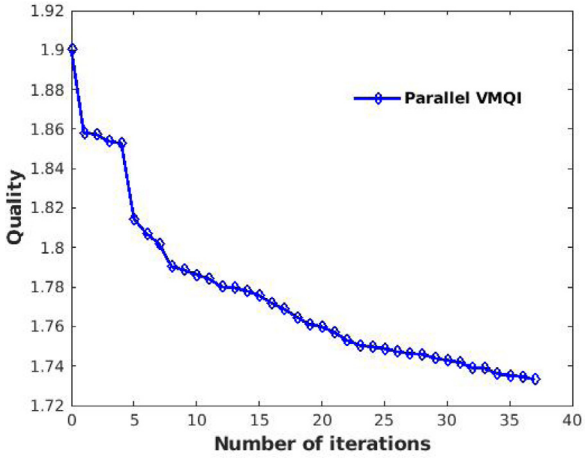


Fig. 22. Average quality versus the number of iterations for the 500K element tetrahedral mesh of the tire domain.

the nodal velocities for each interior node. Whereas each core is able to calculate the nodal velocities for the interior nodes within its region, computing the nodal velocities of the shared nodes requires communication among cores. To do this efficiently, the algorithm first calculates the nodal velocities for elements containing at least one shared node. Then we communicate the partial nodal velocities of the shared nodes using a non-blocking collective instruction to overlap communication with computation of the nodal velocities for the interior nodes. When the number of interior nodes per core is high, a total overlap of communication and computation is achieved. Finally, the algorithm calculates the average quality of the mesh in each iteration and uses this information to terminate the computations when no significant improvement of the average mesh quality is observed.

We tested our parallel variational mesh quality improvement algorithm on three different 3D domains which were discretized using tetrahedral meshes. The results of our numerical experiments show good strong scalability and speedup for the meshes with 80M and 40M elements on up to 128 cores. The efficiency observed in the experimental results is the consequence of the complete overlap of communication and computation when calculating the nodal velocities (see Fig. 14). For the test cases presented in this paper, the major source of overhead occurs in the pre-processing step, i.e., where P_0 distributes the data and identifies the shared nodes. In addition to this, if the number of interior nodes on each core is relatively small compared with the total number of shared nodes, then the communication time among cores increases relative to the runtime. Hence we obtain a performance degradation, as a complete overlap of communication and computation is not possible. The weak scaling results we obtained are typical for unstructured meshes.

There are several possible avenues for future research. First, a different communication strategy may be used to decrease the memory consumption and communication time. For example, a local-blocking communication strategy might decrease the performance for a lesser number of cores but should perform better for a greater number of cores. In addition, a parallel preprocessing step will reduce the runtime and memory consumption for P_0. Another possible avenue for research is the adoption of a different domain decomposition strategy, such as node

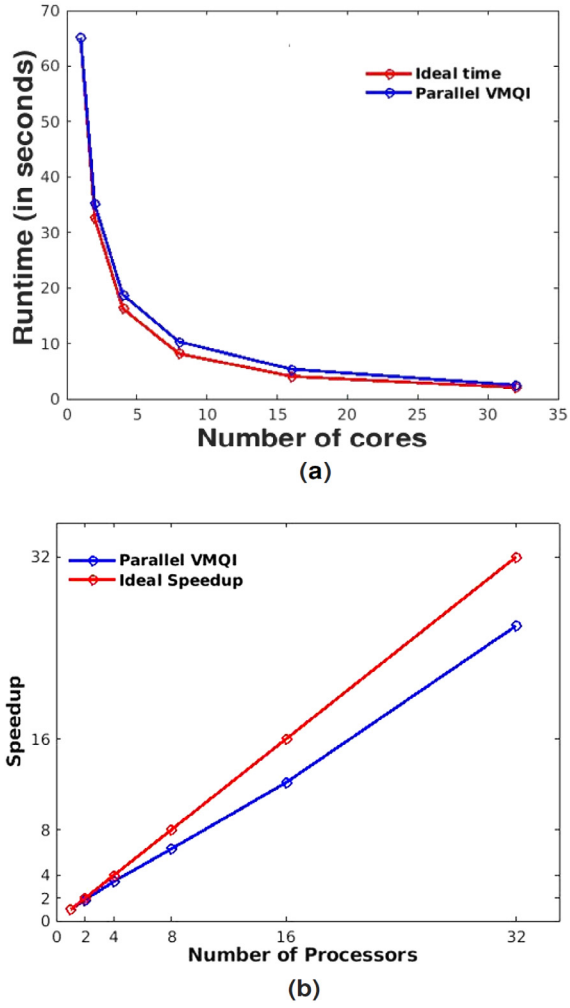


Fig. 23. (a) Total runtime and (b) speedup for the Parallel VMQI algorithm for the 500K element tetrahedral mesh of the tire domain.

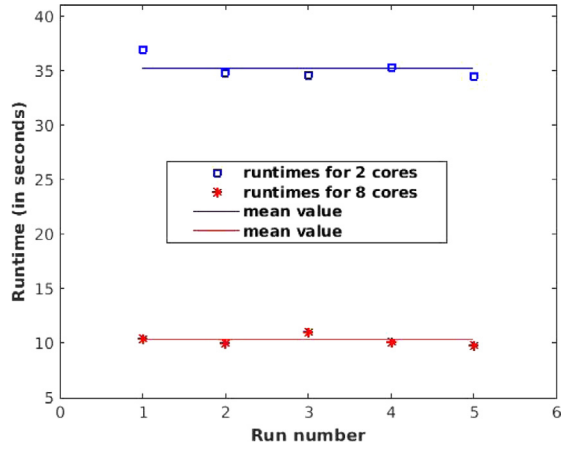


Fig. 24. Wall clock time obtained for 5 runs on 2 and 8 cores. The standard deviations are 0.997 and 0.495 for 2 and 8 cores, respectively.

coloring; this would partition the cores into independent sets. In regard to applications, one can extend the same ideas presented in this paper to variational mesh adaptation algorithms such as the one in [47].

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The work of the first author was supported by National Science Foundation grant OAC-1500487. The work of the second author was supported in part by National Science Foundation grants OAC-1500487 (formerly OAC-1330056 and OAC-1054459), CCF-1717894, and OAC-1808553. The work of all three authors was supported through instrumentation funded by the Army Research Office under contract W911NF-15-1-0377. They also wish to thank the three anonymous referees for their careful reading of the paper and for their helpful suggestions which strengthened it.

References

[1] Southern J, Gorman G, Piggott M, Farrell P. Parallel anisotropic mesh adaptivity with dynamic load balancing for cardiac electrophysiology. J Comput Sci 2012;3:8–16.
[2] Tian F, Dai H, Luo H, Doyle J, Rousseau B. Fluid-structure interaction involving large deformations: 3D simlations and applications to biological systems. J Comput Phys 2014;258:451–69.
[3] Lei J, Wang X, Xie G, Lorenzini G. Turbulent flow field analysis of a jet in cross flow by DNS. J Eng Thermophys 2015;24:259–69.
[4] Aliabadi S, Johnson A, Abedi J, Zellars B. High performance computing of fluid-structure interactions in hydrodynamics applications using unstructured meshes with more than one billion elements. In: Proc. of the 2002 International Conference on High Performance Computing (HiPC 2002), Vol. 2552. Springer Berlin Heidelberg; 2002, p. 519–33.
[5] Mang K, Walloth M, Wick T, Wollner W. Mesh adaptivity for quasi-static phase-field fractures based on a residual-type a posteriori error estimator. GAMM-Mitt 2020;43:e20200000.
[6] Domingues M, Deiterding R, Lopes M, Gomes A, Mendes O, Schneider K. Wavelet-based parallel dynamic mesh adaptation for magnetohydrodynamics in the AMROC framework. Comput Fluids 2019;190:374–81.
[7] Chrisochoides N. A survey of parallel mesh generation methods. In: Bruaset A, Tveito A, editors. Numerical solution of partial differential equations on parallel computers. Springer Berlin Heidelberg; 2006, p. 237–64.
[8] Ito Y, Shih A, Erukala A. Parallel unstructured mesh generation by an advancing front method. Math Comput Simulation 2007;75:200–9.
[9] Chrisochoides N, Chernikov A, Kot A, Linardakis L, Foteinos P. Towards exasale parallel delaunay mesh generation. In: Proc. of the 18th international meshing roundtable. Springer; 2009, p. 319–36.
[10] Ray N, Grindeanu I, Zhao X, Mahadevan V, Jiao X. Array-based hierarchical mesh generation in parallel. In: Proc. of the 24th international meshing roundtable, Vol. 124. Procedia Engineering; 2015, p. 291–303.
[11] Wang X, Jin X, Kou D. A parallel approach for the generation of unstructured meshes with billions of elements on distributed memory supercomputers. Int J Parallel Program 2016;1–31.
[12] Marot C, Pellerin J, Remacle J. One machine, one minute, three billion tetrahedra. Internat J Numer Methods Engrg 2019;117:967–90.
[13] Freitag L, Jones M, Plassmann P. A parallel algorithm for mesh smoothing. SIAM J Sci Comput 1999;20:2023–40.
[14] Jiao X, Alexander P. Parallel feature-preserving mesh smoothing. In: Computational science and its applications – ICCSA, Vol. 3483. 2005, p. 1180–9.
[15] Gorman G, Southern J, Farrell P, Piggott M, Rokos G, Kelly P. Hybrid openMP/MPI anisotropic mesh smoothing. Proc Comput Sci 2012;9:1513–22.
[16] Benítez D, Rodríguez E, Escobar J, Montenegro R. Performance evaluation of a parallel algorithm for simultaneous untangling and smoothing of tetrahedral meshes. In *Proc. of the* 23rd *international meshing roundtable*, 2014, p. 579–98.
[17] Sastry S, Shontz S. A parallel log-barrier method for mesh quality improvement and untangling. Eng Comput 2014;30:503–15.
[18] Cheng Z, Shaffer E, Yeh R, Zagaris G, Olson L. Efficient parallel optimization of volume meshes on heterogeneous computing systems. Eng Comput 2017;33:717–26.
[19] Chen J, Zhao D, Zheng Y, Xu Y, Li C, Zheng J. Domain decomposition approach for parallel improvement of tetrahedral meshes. J Parallel Distrib Comput 2017;107:101–13.

[20] Shontz S, Lopez M, Huang W. A parallel variational mesh quality improvement method for tetrahedral meshes. In: Proc. of the 28th international meshing roundtable. Zenodo; 2020, p. 37–49.
[21] Oliker L, Garbow H, Biswas R. Parallel tetrahedral mesh adaptation with dynamic load balancing. Parallel Comput 2000;26:1583–608.
[22] Casagrande A, Leyland P, Formaggia L, Sala M. Parallel mesh adaptation. Ser Adv Math Appl Sci 2005;69:201–12.
[23] Park Y, Kwon O. A parallel unstructured dynamic mesh adaptation algortihm for3D unsteady flows. Internat J Numer Methods Fluids 2005;48:671–90.
[24] Alauzet F, Li X, Seol E, Shephard M. Parallel anisotropic 3D mesh adaptation by mesh modification. Eng Comput 2006;21:247–58.
[25] Park M, Darfomal D.
[26] Zhou M, Xie T, Seol S, Shephard M, Sahni O, Jansen K. Tools to support mesh adaptation on massively parallel computers. Eng Comput 2012;28:287–301.
[27] Lachat C, Dobrynski C, Pellegrini F. Parallel mesh adaptation using parallel graph partitioning. In *Proc. of the 5th European Conference on Computational Mechanics, Vol. 3*, 2014, p. 2612–23.
[28] Loseille A, Menier V, Alauzet F. Parallel generation of large-size adapted meshes. In: Proc. of the 24th international meshing roundtable, Vol. 124. Procedia Engineering; 2015, p. 57–69.
[29] Gorman G, Rokos G, Southern J, Kelly P. Thread-parallel anisotropic mesh adaptation. In: New challenges in grid generation and adaptivity for scientific computing. SEMA SIMAI springer series, Vol. 5, Springer, Cham; 2015, p. 113–37.
[30] Digonnet H, Coupez T, Laure P, Silva L. Massively parallel anisotropic mesh adaptation. Int J High Perform Comput Appl 2017;33:3–24.
[31] Tang J, Cui P, Li B, Zhang Y, Si H. Parallel hybrid mesh adaptation by refinement and coarsening. Graph Models 2020;101084.
[32] Knupp P. Jacobian-weighted elliptic grid generation. SIAM J Sci Comput 1996;17:1475–90.
[33] Buscaglia G, Dari E. Anisotropic mesh optimization and its application in adaptivity. Internat J Numer Methods Engrg 1997;40:4119–36.
[34] Freitag L, Knupp P. Tetrahedral mesh improvement via optimization of the element condition number. Internat J Numer Methods Engrg 2002;53:1377–91.
[35] Escobar J, Rodriguez E, Montenegro R, Montero G, González-Yuste J. Simultaneous untangling and smoothing of tetrahedral meshes. Comput Methods Appl Mech Engrg 2003;192:2775–87.
[36] Bottasso C. Anisotropic mesh adaptation by metric-driven optimization. Internat J Numer Methods Engrg 2004;60:567–639.
[37] Branets L, Carey G. A local cell quality metric and variational grid smoothing algorithm. Eng Comput 2005;21:19–28.
[38] Zhang Y, Hamza A. PDE-based smoothing from 3D mesh quality improvement. In: Electro/info tech IEEE int. conf.. 2006, p. 334–9.
[39] Freitag L, Knupp P, Munson T, Shontz S. A comparison of two optimization methods for mesh quality improvement. Eng Comput 2006;22:61–74.
[40] Munson T. Mesh shape-quality optimization using the inverse mean-ratio metric. Math Program: Ser A B 2007;110:561–90.
[41] Park J, Shontz S. Two derivative-free optimization algorithms for mesh quality improvement. In: Proc. of the 2010 international confernece on computational science, Vol. 1. Procedia Computer Science; 2010, p. 387–96.
[42] Park J, Shontz S. An alternating mesh quality metric scheme for efficient mesh quality improvement. In: Proc. of the 2011 International Conference on Computational Science, Vol. 4. Procedia Computer Science; 2011, p. 292–301.
[43] Sastry S, Shontz S. Performance characterization of nonlinear optimization methods for mesh quality improvement. Eng Comput 2012;28:269–86.
[44] Kim J, Panitanarak T, Shontz S. A multiobjective mesh optimization framework for mesh quality improvement and mesh untangling. Internat J Numer Methods Engrg 2013;94:20–42.
[45] Sastry S, Shontz S, Vavasis S. A log-barrier method for mesh quality improvement and untangling. Eng Comput 2014;30:315–29.
[46] Huang W, Kamenski L, Si H. Mesh smoothing: An MMPDE approach. In: Research note at the 24th international meshing roundtable. 2015.
[47] Huang W, Kamenski L. A geometric discretization and a simple implementation for variational mesh generation and adaptation. J Comput Phys 2015;301:322–37.
[48] Fabritius B, Tabor G. Improving the quality of finite volume meshes through genetic optimisation. Eng Comput 2016;32:425–40.
[49] Shang M, Zhu C, Chen J, Xiao Z, Zheng Y. A parallel local reconnection approach for tetrahedral improvement. In: Proc. of the 25th international meshing roundtable, Vol. 163. Procedia Engineering; 2016, p. 289–301.
[50] Zint D, Grosso R. Discrete mesh optimization on GPU. In: Proc. of the 27th international meshing roundtable. Lecture notes in computational science and engineering, Vol. 127, Springer, Cham; 2019, p. 445–60.
[51] Huang W, Ren Y, Russell R. Moving mesh partial differential equations (MMPDEs) based upon the equidistribution principle. SIAM J Numer Anal 1994;31:709–30.
[52] Huang W, Kamenski L, Russell R. A comparative study of meshing functionals for variational mesh adaptation. J Math Study 2015;48:168–86.
[53] Alliez P, Cohen-Steiner D, Yvinec M, Desbrun M. Variational tetrahedral meshing. ACM Trans Graph 2005;24:617–25.
[54] Hachem E, Feghali S, Codina R, Coupez T. Anisotropic adaptive meshing and monolithic variational multiscale method for fluid-structure interaction. Comput Struct 2013;122:88–100.
[55] Ferro N, Micheletti S, Perotto S. Anisotropic mesh adaptation for crack propagation induced by a thermal shock in 2D. Comput Methods Appl Mech Engrg 2018;331:138–58.
[56] Clerici F, Ferro N, Marconi S, Micheletti S, Negrello E, Perotto S. Anisotropic adapted meshes for image segmentation: Application to three-dimensional medical data. SIAM J Imaging Sci 2020;13:2189–212.
[57] de Almeida V. Domain deformation mapping: Application to variational mesh generation. SIAM J Sci Comput 1999;4:1252–75.
[58] Knupp P, Robidoux N. A framework for variational grid generation: Conditioning the Jacobian matrix with matrix norms. SIAM J Sci Comput 2000;21:2029–47.
[59] Liao G. Variational approach to grid generation. Numer Methods PDE 1992;8:143–7.
[60] Winslow A. Adaptive mesh zoning by the equipotential method. Tech. rep. UCID-19062, Lawrence Livermore National Laboratory; 1981.
[61] Brackbill J, Saltzman J. Adaptive zoning for singular problems in two dimensions. J Comput Phys 1982;46:342–68.
[62] Dvinsky A. Adaptive grid generation from harmonic maps on Riemannian manifolds. J Comput Phys 1991;95:450–76.
[63] Huang W. Variational mesh adaptation: Isotropy and equidistribution. J Comput Phys 2001;174:903–24.
[64] Huang W, Russell R. Adaptive moving mesh methods. New York: Springer; 2011.
[65] Huang W, Kamenski L. On the mesh nonsingularity of the moving mesh PDE method. Math Comp 2018;87:1887–911.
[66] Huang W, Kamenski L, Si H. Mesh smoothing: An MMPDE approach. In: Procedia engineering. Proc. of the 24th international meshing roundtable. 2015.
[67] Dassi F, Kamenski L, Farrell P, Si H. Tetrahedral mesh improvement using moving mesh smoothing, lazy searching flips, and RBF surface reconstruction. Comput Aided Des 2018;103:2–3.
[68] Si H. Tetgen: A quality tetrahedral mesh generator and three-dimensional delaunay triangulator. 2019, URL http://wias-berlin.de/software/ (accessed 6/17/2019).
[69] Mathews J, Fink K. Numerical methods using MATLAB. 3rd ed.. Prentice Hall; 1999.
[70] Karypis G, Kumar V. A fast and highly quality multilevel scheme for partitioning irregular graphs. SIAM J Sci Comput 1999;20:359—392.
[71] 3D models, CAD solids - 3D CAD browser. 2001-2019, URL http://www.3dcadbrowser.com/ (accessed 6/17/2019).
[72] GAMMA3: Automatic mesh generation and adaptation methods. 2019, URL https://team.inria.fr/gamma3/ (accessed 6/17/2019).
[73] Meshlab. 2019, URL http://www.meshlab.net (accessed 6/21/2019).