Parallel Performance Model for Vertex Repositioning Algorithms and Application to Mesh Partitioning

D. Benitez, J.M. Escobar, R. Montenegro, E. Rodriguez

Abstract Many mesh optimization applications are based on vertex repositioning algorithms (VrPA). Since the time required for VrPA programs may be large and there is concurrency in processing mesh elements, parallelism has been used to improve performance. In this paper, we propose a performance model for parallel VrPA algorithms that are implemented on memory-distributed computers. This model is validated on two parallel computers and used in a quantitative analysis of performance scalability, load balancing and synchronization and communication overheads. We show that load imbalance and synchronization between boundary partitions are the major causes of the parallel bottlenecks. In order to diminish load imbalance, a new approach to mesh partitioning is proposed. This strategy reduces the imbalance in mesh element evaluations caused by multilevel k-way partitioning algorithms and consequently, improves the performance of parallel VrPA algorithms.

1 Introduction

There are several areas of research involving parallel processing of meshes. For example, many mesh processing techniques have been developed to generate meshes in parallel [5]. The sizes and shapes of generated elements affect the efficiency and accuracy of computational applications. Thus, other parallel algorithms are used for mesh optimization [8]. Additionally, parallel mesh warping algorithms have been developed which employ optimization methods for use in computational simulations with deforming domains [15].

A few performance models for parallel meshing algorithms have been developed. Such models can enable us to understand, fine-tune and predict the performance of applications. Barker and Chrisochoides applied an analytical model for load balancing to mesh generation asynchronous applications [1]. Sarje et al. used a performance model to propose a mesh partitioning that improves the load balancing of an ocean modeling code [16]. Mathis and Kerbyson presented a parametric model to predict the parallel performance of a partial differential equation solver on unstructured meshes [13].

SIANI institute & DIS department, University of Las Palmas de Gran Canaria, Spain.

Vertex repositioning algorithms (VrPA) have been adopted by a vast majority of mesh optimization applications [6, 7, 8, 17, 18], but no performance model for distributed-memory computers has been proposed yet. In another paper, we have proposed a performance model for sequential VrPA algorithms [2]. Using this model, we propose in this paper a performance model for loosely synchronous algorithms executed on distributed-memory computers (Sections 4 and 5). The parallel model was applied to several VrPA algorithms for mesh untangling and smoothing and the results in prediction accuracy are shown in Section 6. Additionally, the parallel model is used in Section 7 to study the performance scalability, load balancing and synchronization and communication overheads of VrPA. Based on the parallel model and the results of its validation, a new approach to mesh partitioning that reduces load imbalance is proposed in Section 8. Before contributions are explained, the following two sections describe a generalized version of parallel VrPA algorithms and implementation details.

2 Generalized parallel algorithm

VrPA algorithms have been used to untangle a mesh and/or improve its quality by moving the free vertices [8, 17, 18]. They can be posed as numerical techniques in which the following parameters are considered [6]: objective function approach (A) and formulation (f), element quality metric (q), minimization method (NM) and convergence or termination criteria (TC). There are many choices for each free parameter one could make in a study of VrPA algorithms. In this paper, we limited the options to those shown in Table 1. Each combination of choices will be called *VrPA configuration* and denoted: $\langle A\rangle$-$\langle f\rangle$-$\langle q\rangle$-$\langle NM\rangle$-$\langle TC\rangle$, for instance, “Lo-D2-hS-CG-TC2”.

Algorithm 1 shows a generalized VrPA for distributed-memory computers that is similar to others [8, 17]. The input is a set of nC files with information of vertices and elements of mesh partitions: $P_i,\ i \in \{1, \ldots, nC\}$. Every partition includes spatial coordinates of vertices and information of element edges. A partitioning tool is used to obtain these files from the file with information of a mesh, M.

The M mesh is frequently partitioned by assigning each vertex to one partition [8, 17]. In this way, the number of send and receive messages between parallel processes is minimized. Each partition is required to additionally include information of all vertices of elements where at least one vertex is assigned to that partition. The boundary of a partition is constituted by shared elements, each of them is formed by vertices assigned to that partition and at least to another partition.

In each mesh partition, vertices are classified as interior, non-ghost boundary (or simply, boundary), ghost or fixed. Interior vertices form elements whose all vertices belong to that partition. Boundary vertices form shared elements where at least one vertex belongs to another partition. Ghost vertices are these vertices that belong to other partitions. Thus, ghost vertices are replicated in shared partitions.

Table 1: Free VrPA parameters and their choices that are considered in this paper. Legend: RW denotes related work.

Parameter	Options		RW
Objective function approach (A): $K = \sum_{i=1}^{n} f(q_i)$ n: total free elements*	**Gl**: All-vertex (K: Global function)	$n = N_M$: free elements* of mesh	[6]
	Lo: Single-vertex (K: Local function)	$n = N_v$: free elements* of local patch	[6]
Objective function formulation: $f(q_i)$ q_i: quality of i^{th} element $q_{min} = min(q_i)_{i \in \{1 \dots n\}}$ $h(z) = \frac{1}{2}\left(z + \sqrt{z^2 + 4\delta^2}\right)$ $\delta, \mu, \tau =$ constants	**D1**: Distortion 1	$f(q_i) = q_i^{-1}$	[4]
	D2: Distortion 2	$f(q_i) = q_i^{-2}$	[4]
	log1: Logarithmic barrier 1	$f(q_i) = n^{-1} q_{min}^{-1} - \mu\ log(\tau\ q_{min}^{-1} - q_i^{-1})$	[2]
	inv: Regularized barrier	$f(q_i) = q_i^{-1} + \frac{1}{h\left(q_{min}^{-1} - q_i^{-1}\right)}$	[2]
Element quality metric: q_i S_i: Jacobian matrix $\|\| \ \|\|_F$: Frobenius norm	**hS**: Regularized mean ratio $h(z) = \frac{1}{2}\left(z + \sqrt{z^2 + 4\delta^2}\right)$ $\sigma_i = determinant(S_i)$	$q_i = \frac{d\ [h(\sigma_i)]^{2/d}}{\left\|\left\|S_i\right\|\right\|_F^2}$ $triangle: d = 2, s = 3$ $tetrahedron: d = 3, s = 6$ $a, b, \delta, \lambda = constants$	[7]
Numerical minimization method (NM)	**CG**: Conjugate Gradient	Polack-Ribiere, analytical derivatives	[4]
	SD: Steepest Descent	Analytical derivatives	[4]
Termination criteria (TC) Q_i: mean-ratio quality value of the i^{th} element $Q_{min} = min(Q_i)_{i \in \{1 \dots N_M\}}$	**TC2** (optimum mesh) $\overline{Q}$: average mean-ratio value of mesh Δ: maximum variation between outer iterations	$true = (\ Q_{min} > 0\ and\ \Delta\overline{Q} < 10^{-3}\ and\ \Delta Q_{min} < 10^{-3}\ )$	

* *Free element*: mesh element with at least one free vertex.

Each partition is assigned to a different parallel process that optimizes interior and boundary vertices but not ghost vertices. The numerical processing is divided into three parallel phases. The first phase is implemented in lines 26 to 32. It is used only once to prepare the processing of vertices laying on the partition boundaries in phase 3.

When a boundary vertex is being repositioned in phase 3, the numerical method needs the coordinates of all connected vertices that should remain fixed. Computational dependency appears between the adjacent boundary and ghost vertices because one vertex begins to be processed after another has been repositioned. Thus, vertices of shared elements cannot be optimized in parallel.

BoundaryColoring divides the boundary of a partition (P_i) into nF independent sets (I_{ij}), also called *colors* (line 28) [3]. After that, the order of processing and interchange of boundary and ghost vertices is established. The resulting orderings are interchanged among shared partitions (line 29).

Algorithm 1 - Parallel mesh vertex repositioning algorithm.

```
▷ Input: files with information of $P_i$ partitions, $P_i \leftarrow \mathit{Partition(M)}, i \in \{1,\ldots,nC\}$
#define: approach ($A$), formulation ($f$), quality metric ($q$), numerical minimization method ($NM$)
#define termination criteria: $TC = \mathit{LogicFunction}(Q_{min}, \Delta\overline{Q}, \Delta Q_{min})$
#define constants: $\tau = 10^{-6}$ (maximum or minimum increase of the objective function), $N_{mII} = 150$ (maximum number of inner iterations), $N_{mOI} = 100$ (maximum number of outer iterations)
$N_{e,i} \leftarrow 0$    ▷ element evaluations for partitions $P_i, i \in \{1,\ldots,nC\}$
procedure VertexRepositioning($W, X, n$)
▷$\mathit{Inputs}: W$(free vertices),$X$(their coordinates),$n$(number of elements)
    ▷ $\mathit{Initiation}:\ K = 0,\ \Delta K = 0,\ m = 0$ (inner loop index)
    while ($\Delta K \le \tau$(minimizing) or $\Delta K \ge \tau$(maximizing)) and $m \le N_{mII}$ do    ▷ Inner loop
        $\hat{X} \leftarrow X$    ▷ Returned spatial coordinates ($\hat{X}$) of vertices ($W$)
        ▷ $\mathit{Initiation}:\ P \leftarrow 0$    ▷ Moving directions: $P = \{p_v\}, v \in W$
        for $i = 1,\ldots,n$ do    ▷ $n$: number of free elements
            for each free vertex $v$ of $i^{th}$ free element do
                $p_v$ += NM($f'(q_i)$,$v$)    ▷ $f'$: derivatives used in NM
                $N_e$ += 1    ▷ Number of mesh element evaluations
        $X \leftarrow \hat{X} + P$    ▷ Tentative positions of free vertices
        $K_t \leftarrow 0$    ▷ Initial value of objective function
        for $i = 1,\ldots,n$ do
            $K_t$ += $f(q_i)$    ▷ MESH ELEMENT EVALUATION
            $N_e$ += 1    ▷ Number of mesh element evaluations
        $\Delta K \leftarrow K_t - K$
        $K \leftarrow K_t$    ▷ Final value of objective function
        $m$ += 1    ▷ Number of inner iterations
    return $\hat{X}$    ▷ Output: updated coordinates of free vertices
procedure main( )
    for $P_i \in M$ in parallel do    ▷ Parallel phase 1: begin
        ▷ Read the vertex and element information of $P_i$ partition
        $I_i \leftarrow$ BoundaryColoring($P_i$)    ▷ $I_i = \{I_{ij}\}_{j \in \{1..nF_i\}, i \in \{1..nC\}}$
        MPI_Send-MPI_Receive information of boundary/ghost vertices
        ▷ Store the order of partition free boundary/ghost vertices
        $Q_{min,i} \leftarrow$ GlobalMeasures($P_i$)    ▷ Initial partition quality
    Synchronization MPI_Allreduce    ▷ Parallel phase 1: end
    for $P_i \in M$ in parallel do
        ▷ $\mathit{Initiation}:\ \Delta\overline{Q_i} = 10^6,\ \Delta Q_{min,i} = 10^6,\ k_i = 0$ (loop index)
        while $TC \ne true$ and $k_i \le N_{mOI}$ do    ▷ Mesh/Outer loop
            if $A = Gl$ then    ▷ Par. pha. 2 - Interior processing: begin
                $X_V \leftarrow$ VertexRepositioning($V, X_V, N_M$)
            else    ▷ $A = Lo$ (single-vertex)
                for each free interior vertex $v \in P_i$ do
                    $x_v \leftarrow$ VertexRepositioning($v, x_v, N_v$)
            Synchronization MPI_Barrier    ▷ Parallel phase 2: end
            for each boundary independent-set $I_{ij} \in P_i$ do    ▷ Parallel phase 3: begin
                for each free boundary vertex of partition $v \in I_{ij}$ do
                    $x_v \leftarrow$ VertexRepositioning($v, x_v, N_v$)
                MPI_Send-MPI_Receive coordinates of vertices $x_v$
            Synchronization MPI_Barrier    ▷ All boundary vertices
            $(Q_{min,i}, \Delta\overline{Q_i}, \Delta Q_{min,i}) \leftarrow$ GlobalMeasures($P_i$)
            $k_i$ += 1    ▷ Number of mesh/outer iterations
            Synchronization MPI_Allreduce    ▷ Parallel phase 3: end
        MPI_Send-MPI_Receive $N_{e,i}$
    ▷ Output: files with information of optimized $P_i$ partitions
```

Finally, a list with the order of boundary and ghost vertices is created in line 30. This list determines the order in which these vertices are optimized in the parallel process, or received from other processes in phase 3. Interior vertices do not need to be reordered because all adjacent vertices are assigned to the same process. Using the function `MPI_Allreduce` at the end of phase1, a synchronization barrier ensures that all partitions have completed these steps before continuing computation (line 32).

Mainly, the algorithm spends most of the time in a variable number of concurrent partition sweeps (lines 35-50). In each *partition sweep*, also called *outer* or *mesh iteration*, all interior and boundary vertices are optimized separately in parallel phases 2 (lines 36. . . 41) and 3 (lines 42. . . 49), respectively, adjusting the spatial coordinates of all free vertices (X_V). The vertices that lie on the mesh surface are treated as fixed and are not updated.

Interior vertices of every partition are not dependent on vertices of other partitions and are sequentially optimized by the same parallel process. In this way, the interior vertices of all partitions are optimized concurrently. A single synchronization phase between partitions is established to ensure that all interior vertices are completely repositioned (line 41).

For partition boundaries, some independent sets of free vertices from different partitions (I_{ij}) are optimized in parallel. After an independent set has been optimized, the interchange of updated coordinates is implemented using message send/receive functions (line 45). These computation-synchronization-communication phases are repeated until all boundary vertices have been optimized (lines 42. . . 45).

When all boundary vertices have been updated (line 46), the minimum quality metrics of all partitions are calculated and distributed (line 47. . . 49) and the partition sweep finishes. *GlobalMeasures* provides the average and minimum mean-ratio quality metric of the mesh [6]. At this moment, a new partition sweep may begin if convergence conditions are not met by every partition (line 35). *LogicFunction* uses termination criteria (TC) to stop the algorithm. The *outer loop* is iterated in $Main$ procedure while *LogicFunction* is not true. After a variable number of partition sweeps, the output of our parallel algorithm provides optimized mesh partitions (line 51).

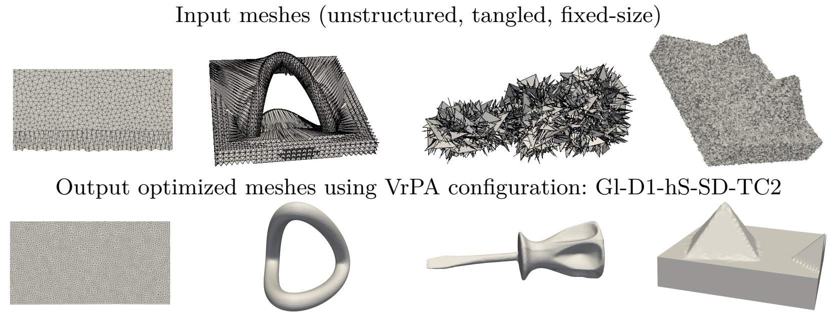
The most time-consuming operation is a sequential procedure called $VertexRepositioning$ (lines 37, 40, 44), which moves free vertices (V) of an input mesh partition (P_i). Thus, the parallel performance is based on the performance of the underlying serial numerical method. $VertexRepositioning$ iterates an *inner loop* while an extreme of the objective function (K) is being reached by using an optimization solver (NM). K is constructed with A, f and q as it is shown in Table 1.

3 Experimental setup

Software framework. We developed programs for simultaneous mesh untangling and smoothing that implement Algorithm 1. They include double-precision floating-point data structures and functions from *MPI* and the *Mesquite* C++ library [4], which is specialized in mesh smoothing. *Mesquite* was extended to support **hS** quality metric, **log1** and **inv** objective function formulations and **TC2** termination criteria (see Table 1). TC2 is met when the output mesh has no inverted elements and is optimum, i.e., the minimum and average values of the mean-ratio quality metric in two successive mesh iterations do not change significantly. We used *OpenMPI* 1.6.5 and *gcc* 4.8.4 with `-O2` flag on *Linux* systems. A pure sequential version was selected for baseline runs. For each VrPA configuration, we repeated the execution of the sequential and parallel programs several times, such that the 95% confidence interval was lower than 1%.

Benchmark meshes. Algorithm 1 was applied on the unstructured, fixed-sized meshes shown in Figure 1 whose characteristics are in Table 2. The 2D mesh was obtained by using *Gmsh* tool [9], taking a square, meshing with triangles and displacing selected nodes of the boundary. This type of tangled mesh can be found in some problems with evolving domains [12]. All 3D meshes were obtained from a tool for adaptive tetrahedral mesh generation that tangles the mesh [14]. All the mesh sizes were always fixed, and we used *Metis* 5.1.0 for mesh partitioning [11].

Platforms. Numerical experiments were conducted on two cluster computers called *Cluster1* and *Cluster2* that are in two different locations. *Cluster1* is a Bull computer with 28 compute nodes that are organized in 7 BullxR424E2 servers. They are interconnected with Infiniband QDR 4X (32 Gbit/s). Each node integrates two Intel Xeon E5645 (6 cores each, 2.4 GHz), and 48 GB of DDR3/1333. So, up to 336 cores were used in parallel. The storage system is a RAID-5 disk array consisting of 7200 RPM SATA2 disk drives. All compute nodes share a common file system through NFS


Input meshes (unstructured, tangled, fixed-size)
Output optimized meshes using VrPA configuration: Gl-D1-hS-SD-TC2

(a) Square(2D) (b) Toroid(3D) (c) Screwdr.(3D) (d) Egypt(3D)

Fig. 1: Input and output meshes for four optimization problems solved with the same VrPA algorithm.

Table 2: Characteristics of input meshes. All meshes have inverted elements: Q_{min}=0.

Mesh characteristic	**Square**	**Toroid**	**Screwdriver**	**Egypt**
Total vertices	3314499	9176	39617	1724456
Free vertices (they can be moved)	3309498	3992	21131	1616442
Fixed vertices (they are not moved)	5001	5184	18486	108014
Element type: triangle (2D), tetrahedron (3D)	2D	3D	3D	3D
Total free elements (N_M)	6620936	35920	168834	10013858
Inverted/Tangled elements (%)	0.1%	38.2%	49.4%	46.2%
Average mean-ratio quality metric ($\overline{Q}$)	0.95	0.17	0.13	0.23
Standard deviation of the mean-ratio metric	0.05	0.31	0.21	0.27

over a gigabit Ethernet LAN. *Cluster2* is a Fujitsu computer that has the same type of network, storage and file system as *Cluster1* but only four compute nodes (Primergy CX250) with 16 E5-2670@2.6GHz cores and 32 GB of DDR3/1600 per node. We activated multiples of 12 or 16 cores to completely occupy the compute nodes. During the experiments, the compute nodes were not shared among other user-level workloads. Additionally, multithreading and Turbo Boost were disabled.

4 Sequential performance model

In another paper, we have proposed a performance model for sequential VrPA algorithms that tries to justify their execution times [2],

$$t_{CPU}^{Smodel} = \alpha \; N_e \tag{1}$$

with t_{CPU}^{Smodel} the execution time, N_e the number of mesh element evaluations and α the model parameter that represents the time per element evaluation. Equation 1 assumes that computation time is much larger than total input/output time.

N_e takes into account multiple evaluations of an element quality metric and its derivative (see lines 15 and 20 in Algorithm 1). Although not exactly equal, N_e is similar to the *concurrent function evaluation steps* defined in [19]. We have demonstrated that the execution time of VrPA algorithms is more proportional to N_e than other workload measures such as mesh size or objective function evaluations (see [2] for a more extended discussion). It is important to note that N_e depends not only on the problem size but also on the number of inner and outer iterations required to meet the convergence criteria. However, N_e is independent of the computer hardware; it depends on the algorithm and its implementation, the selected numerical accuracy of data structures, and the method chosen by the compiler to implement arithmetic operations. We use the number of element evaluations as workload measure in the new parallel performance model that is presented below.

The other factor, the time per element evaluation (α), is more affected by the processor and objective function approach than the objective function formulation, the quality metric, numerical solver, convergence criteria or mesh [2]. Thus, to precisely determine execution times with Equation 1, α must be recalculated when the processor and algorithm configuration change.

In summary, this simple performance model determines that the time required by sequential VrPA algorithms to optimize a mesh is directly proportional to the number of element evaluations.

5 Parallel performance model

In this section, we describe a new model to justify the parallel runtimes of Algorithm 1 for a selected VrPA configuration on a determined distributed-memory computer. This model uses the time per mesh element evaluation (α), which is obtained from the sequential execution, denoted *Sreal*, of the same configuration using a pure sequential version of Algorithm 1. Then, selecting one VrPA configuration and employing Equation 1,

$$\alpha = \frac{t_{CPU}^{Sreal}}{N_e} \tag{2}$$

We assume that this model parameter is constant for all parallel experiments that use the same VrPA configuration, mesh and cluster computer.

Since there is an MPI barrier between the repositioning of interior and partition boundary vertices (line 41), Equation 3 models the parallel execution time that is divided into two components, one for optimizing interior vertices and the other for partition boundary vertices,

$$t_{CPU}^{Pmodel} = t_{interior}^{Pmodel} + t_{boundary}^{Pmodel} \tag{3}$$

In this case, we have assumed that the execution time for the mesh partitioning phase and parallel phase 1 are negligible with respect to parallel phases 2 and 3. The parallel time for interior vertices is expressed as a sum,

$$t_{interior}^{Pmodel} = t_{scalable,interior}^{Pmodel} + t_{imbalance,interior}^{Pmodel} \tag{4}$$

where the first term denotes the *scalable interior parallelism*. If the workload was evenly distributed among nC partitions, the total workload for optimizing interior vertices in all partitions ($N_{e,interior}^{Pmodel}$) would be divided by nC,

$$t_{scalable,interior}^{Pmodel} = \alpha \, \frac{N_{e,interior}^{Pmodel}}{nC} \tag{5}$$

The second component of Equation 4, called *interior imbalance*, measures the additional time required by the most loaded partition when the workload for processing interior vertices is not evenly distributed. It is given by Equation 6, where $N_{e,interior,max}^{Pmodel}$ is the maximum number of interior element evaluations of a partition,

$$t_{imbalance,interior}^{Pmodel} = \alpha \left(N_{e,interior,max}^{Pmodel} - \frac{N_{e,interior}^{Pmodel}}{nC} \right) \tag{6}$$

The values of $N^{Pmodel}_{e,interior}$ and $N^{Pmodel}_{e,interior,max}$ for Equations 5 and 6 are measured at the end of the parallel execution. The time needed to optimize all partition boundary vertices has four terms,

$$t^{Pmodel}_{boundary} = t^{Pmodel}_{scalable,boundary} + t^{Pmodel}_{imbalance,boundary} + t^{Pmodel}_{synchro,boundary} + t^{Pmodel}_{comm,boundary} \tag{7}$$

Scalable boundary parallelism (Equation 8) assumes that the workload of boundary vertices ($N^{Pmodel}_{e,boundary}$) are evenly distributed among nC partitions.

$$t^{Pmodel}_{scalable,boundary} = \alpha \, \frac{N^{Pmodel}_{e,boundary}}{nC} \tag{8}$$

Boundary imbalance (Equation 9) measures the additional time needed by the most loaded partitions when workloads of the nF independent sets are not evenly distributed,

$$t^{Pmodel}_{imbalance,boundary} = \alpha \Big(\sum_{j=1}^{nF} N^{Pmodel}_{e,boundary,j,max} - \frac{N^{Pmodel}_{e,boundary}}{nC} \Big) \tag{9}$$

where $N^{Pmodel}_{e,boundary,j,max}$ is the maximum number of element evaluations of the j^{th} independent-set of a partition. $N^{Pmodel}_{e,boundary}$ and the accumulated value ($\sum N^{Pmodel}_{e,boundary,j,max}$) in Equations 8 and 9 are obtained at the end of parallel execution for a given number of partitions (nC).

Synchronization (Equation 10) assumes that all partitions cannot optimize boundary vertices concurrently in phase 3. It is due to the vertex dependence imposed by the processing and interchange order of boundary and ghost vertices that is determined in phase 1. In this term of the model, the scalable workload of boundary processing is factored with $(nC - nC')/nC'$, where nC' is the number of partitions that actually are optimizing vertices concurrently in phase 3 ($0 < nC' \leq nC$). As fewer opportunities for parallelism are available in boundary phase, nC' will reduce and the modeled effect causes an increase in execution time. nC' is obtained by averaging the number of partitions that finish a vertex reposition between another partition terminates two consecutive repositioning of boundary vertices.

$$t^{Pmodel}_{synchro,boundary} = \frac{\alpha \, N^{Pmodel}_{e,boundary}}{nC} \, \frac{nC - nC'}{nC'} \tag{10}$$

Equation 11 measures the MPI communication overhead of boundary processing using a two-parameter model for SMP nodes working in the short regime [10]. This equation has two terms times the number of outer iterations (k). The first term represents the *communication latency*, which is modeled as the network latency (LAT) times the number of data block communications ($2\ \beta\ nF$) during a single outer iteration. β denotes the total number of edges of a new graph that represents which partitions share boundary elements. An edge represents the boundary between two partitions. Neighboring partitions are represented by adjacent vertices. An MPI communication is performed through each edge of this graph after processing an independent set. nF denotes the average number of independent sets per partition that is obtained in phase 1.

$$t_{comm,boundary}^{Pmodel} = k \left(LAT \ 2 \ \beta \ nF \ + \ \frac{32 \ (\gamma - 1) \ nV_{boundary}}{BW} \right) \tag{11}$$

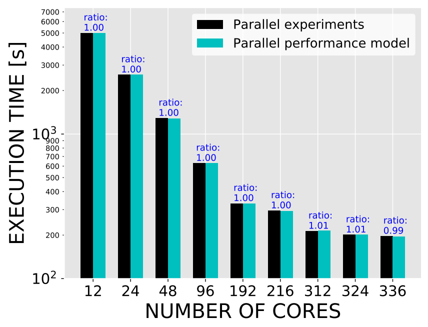
The other term is *data transmission* time, where BW denotes the data rate that each process can achieve in sending or receiving a message. The effective rate is dependent on transmitted data size. However, we assume this parameter is constant because the variability of message sizes is small and computing time significantly exceeds communication time. Each vertex has a data size of 32 bytes to send/receive spatial coordinates and global ID. γ denotes the average number of partitions that share the same vertex, and $nV_{boundary}$ the total number of free boundary vertices of all partitions. Using code instrumentation, we measured LAT and BW in each MPI process and their average values were used to determine the parameters in our performance model. The rest of parameters, β, γ and $nV_{boundary}$, are obtained from the partitions of the input mesh at the end of partitioning phase.

6 Validation of the parallel model

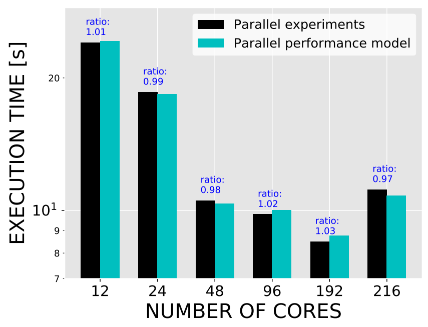
Algorithm 1 was applied to four mesh optimization problems using different VrPA configurations but fixing the TC2 convergence criteria. TC2 is met when the mesh is untangled and smoothed. In order to demonstrate the applicability of our parallel model to a variety of VrPA configurations, a different one was selected for each mesh. The values used for the α model parameter were calculated with Equation 2 and are shown in Table 3.

Figures 2 and 3 depict results that were obtained from *Clusters 1* and *2*, respectively. The resulting execution times (t_{CPU}^{Preal}) are compared to the predictions of our parallel model (t_{CPU}^{Pmodel}). In these tests, the numbers of partitions, MPI processes and CPU cores had the same value. We include results obtained using partitions that activated all cores of different subsets of compute nodes. Thus, each bar diagram shows execution times for numbers of cores that are multiple of 12 (*Cluster1*) or 16 (*Cluster2*). For each mesh optimization problem, note in Table 3 that the minimum qualities of output meshes are similar.

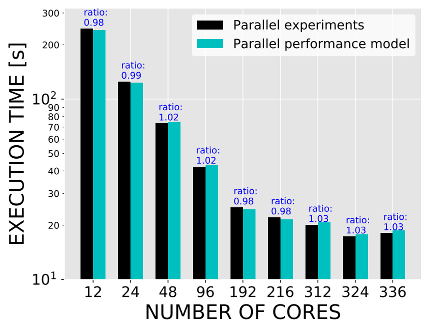
On average, the mean relative errors of our parallel model in the estimation of the times obtained from *Cluster1* and *Cluster2* were 0.027 and 0.031, respectively. This discrepancy can be explained by the inaccuracy introduced when nC' and $\sum_j N_{e,boundary,j,max}^{Pmodel}$ were obtained. Another source of inaccuracy is introduced by α that may be slightly different between parallel and sequential processing.



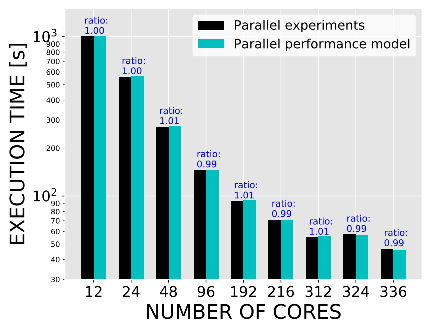
(a) **Square**



(b) **Toroid**



(c) **Screwdriver**



(d) **Egypt**

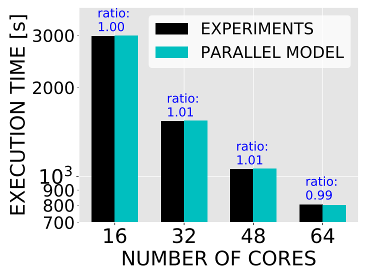
Fig. 2: Results of the parallel experiments and performance model using *Cluster1*. The ratio of the performance model to experiment execution time is shown in blue. Table 3 shows the VrPA configurations that were applied and the average values of the minimum mean-ratio quality metric of output meshes. Performance of the sequential executions of respective configurations is also shown in Table 3.

Table 3: Performance of the baseline sequential experiments and average values of the minimum mean-ratio quality metric ($\overline{Q_{min}}$) of the output meshes of parallel experiments whose results are shown in Figures 2 and 3. α denotes the time per mesh element evaluation that was employed when the parallel model was applied.

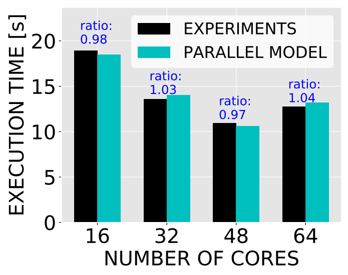
Mesh	**VrPA configuration**	**Sequential experiments**				**Parallel experiments**	
		CPU time [sec]		α $[\mu sec/element]$		$\overline{Q_{min}}$	
		Cluster1	**Cluster2**	**Cluster1**	**Cluster2**	**Cluster1**	**Cluster2**
Square	Lo-D2-hS-SD-TC2	$5.8\ 10^4$	$5\ 10^4$	0.5	0.4	0.633 ± 0.001	0.633 ± 0.001
Toroid	Gl-inv-hS-SD-TC2	19.5	11.0	1.1	0.7	0.333 ± 0.094	0.318 ± 0.108
Screwdriver	Gl-log1-hS-CG-TC2	$1.6\ 10^3$	$1.0\ 10^3$	0.8	0.6	0.255 ± 0.002	0.256 ± 0.001
Egypt	Lo-D1-hS-SD-TC2	$1.1\ 10^4$	$9.0\ 10^3$	0.5	0.4	0.201 ± 0.002	0.202 ± 0.002

7 Parallel performance analysis

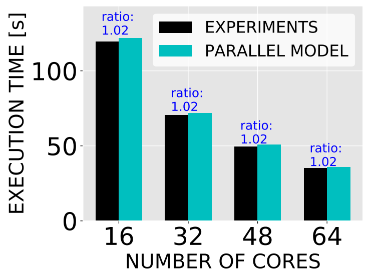
Figure 4 shows stacked column graphs for the times provided by our parallel model when *Cluster1* was used to run Algorithm 1. Every single column corresponds to a determined VrPA configuration, benchmark mesh and number



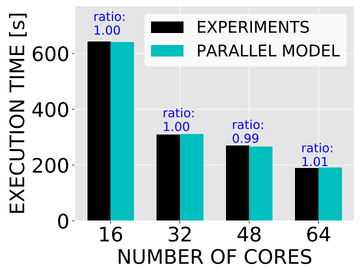
(a) **Square**



(b) **Toroid**



(c) **Screwdriver**



(d) **Egypt**

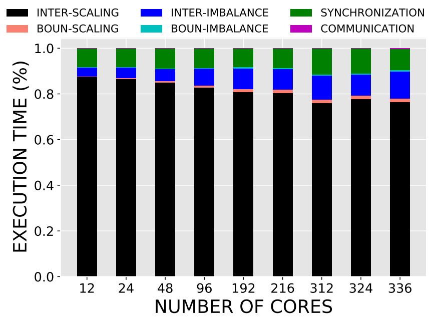
Fig. 3: Results of the parallel experiments and performance model using *Cluster2*. The ratio of the performance model to experiment execution time is shown in blue. Table 3 shows the VrPA configurations that were applied and the average values of the minimum mean-ratio quality metric of output meshes. Performance of the sequential executions of respective configurations is also shown in Table 3.

of partitions. It is divided into six sections, which are grouped into four categories: *scalable parallelism*, *imbalance*, *synchronization* and *communication*.

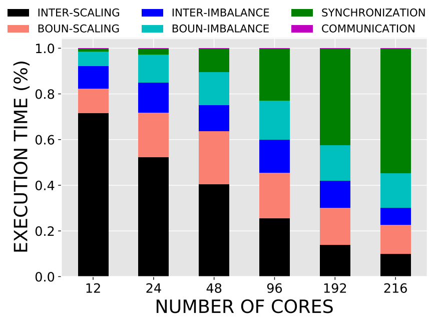
Scalable parallelism includes runtimes for optimizing interior and boundary vertices if the sequential workloads were evenly distributed over all mesh partitions (Equations 5 and 8). These times are represented in Figure 4 by the two bottom columns denoted as *Inter-Scaling* and *Boun-Scaling*, respectively.

As the number of partitions increases in strong scaling when a mesh is optimized, the time devoted to this category reduces. Equations 5 and 8 predict that it is due to the fact that we are solving fixed-size problems and the element evaluations in each partition reduce. Note in Figure 4 that the fraction of time in scalable parallelism also reduces, which means that overheads are more relevant. However, the fraction of time in boundary optimization tends to increase because the ratio of boundary to interior element evaluations increases when the number of partitions increases.

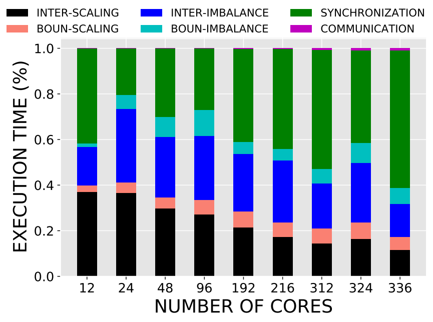
Another increasing trend is observed when problems of different sizes are compared for a given number of cores. For example, using 324 cores in *Cluster1*, note that the fraction of time in scalable parallelism is 25% for Screwdriver mesh, 68% for Egypt mesh and 78% for the Square mesh. Since for 324



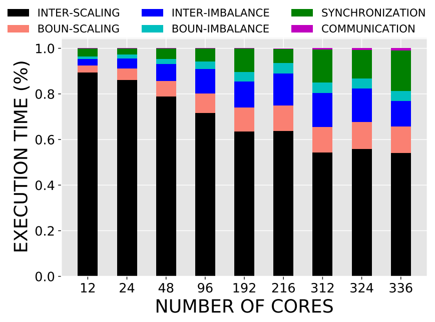
(a) Mesh: Square, VrPA configuration: Lo-D2-hS-SD-TC2



(b) Mesh: Toroid, VrPA configuration: Gl-inv-hS-SD-TC2



(c) Mesh: Screwdriver, VrPA configuration: Gl-log1-hS-CG-TC2



(d) Mesh: Egypt, VrPA configuration: Lo-D1-hS-SD-TC2

Fig. 4: Time breakdowns provided by the parallel performance model for *Cluster*1.

cores Screwdriver requires fewer element evaluations than Egypt and Egypt fewer element evaluations than Square (1.8 10^9, 2.5 10^{10}, 1.1 10^{11}, respectively), the workload distributed among partitions is lower for Screwdriver than for Egypt, which is lower than for Square. Thus, VrPA algorithms cannot compensate for the parallel overheads when Screwdriver is optimized as much as when Egypt or Square are optimized. In general, this performance category is associated with parallel efficiency, which depends mainly on the fraction of time occupied by mesh element evaluations perfectly balanced.

Load imbalance (Equations 6 and 9) is another category that includes the execution times due to processor overload during vertex repositioning when the element evaluations are not well balanced (see *Inter-Imbalance* and *Boun-Imbalance* in Figure 4). Although the load imbalance cost decreases as the number of partitions for a given problem increase because the elements evaluations per partition decrease, its percentage relevance tends to be larger. It is due to the less homogeneous distribution of workload that is assigned by the mesh partitioning tool. Note that this tool distributes vertices and elements among partitions but it does not know in advance how many mesh elements evaluations will be completed. For our largest problems, Square and Egypt, this overhead category is the major cause of the parallel bottlenecks.

For example, using 324 cores in *Cluster1*, load imbalance is responsible for 11% and 19% of the total runtime, respectively.

Synchronization (Equation 10) includes the overheads caused by the independent sets of partition boundary vertices that have to be processed in the order determined in parallel phase 1. Note in Figure 4 that, as the number of partitions (nC) increases, the percentage relevance of this category tends to be larger in all of our optimization problems. It is due to the number of processes that concurrently reposition partition boundary vertices (nC'), which increases less than the number of partitions. This percentage relevance is also affected by the increasing ratio of boundary to interior element evaluations, which is larger as described above when the number of partitions increases.

Another effect of synchronization overhead can be observed when problems of different sizes are compared for a given number of partitions. Taking any number of partitions, the percentage relevance of this category is larger for Toroid and Screwdriver than Square and Egypt. It is due to that nC' tends to reduce when the number of boundary element evaluations reduces. So, the factor of our model $(nC - nC')/nC'$ is larger. The modeled effect is concordant with fewer opportunities for parallelism when the concurrent boundary element evaluations reduce. Moreover, although the number of boundary elements evaluations is smaller in Toroid and Screwdriver than Square and Egypt for a given number of partitions, the ratio of boundary to interior element evaluations is larger in Toroid and Screwdriver than Square and Egypt. Thus, the percentage relevance of synchronization is also larger.

Communication is a category that considers the overhead caused by the transmission of updated coordinates of partition boundary vertices (Equation 11). This overhead increases with the number of partitions because it depends on the numbers of boundary vertices and independent sets. However, its percentage relevance is the lowest, from 1% to 2% when 336 cores are used (see Figure 4). Therefore, VrPA algorithms do not suffer significantly from the MPI communication overhead in our experiments. This is due to the dependence of communication time on the numbers of partition boundary vertices and independent sets, in contrast to the optimization time of boundary vertices and other overhead categories that are dependent on the concurrent mesh element evaluations.

8 Application to load balancing

Mesh optimization algorithms for distributed-memory computers use a previous phase of mesh partitioning to balance and distribute vertices among parallel processes [17]. As stated in Section 3, we used a Metis program (`gpmetis`) based on the multilevel k-way graph partitioning algorithm with edge cut minimization [11]. This program requires as input a file storing a mesh. Part of this file contains information relevant for vertices.

The results of previous section show that load imbalance is a significant parallel overhead. To reduce this overhead, we propose to include in the input file of the partitioning program a weight associated with each vertex. This weight coincides with the number of element evaluations that are needed by the vertex in a previous outer iteration of the parallel execution of the VrPA algorithm. Thus, the first outer iteration is repeated twice, one for weight calculation and the other for mesh optimization. Without vertex weights, the partitioning program balances vertices. With our proposal, this program balances element evaluations, e.g., the sum of evaluations of the vertices assigned to each parallel process is approximately the same across the partitions.

8.1 Hypothesis

Figure 5 shows the mesh element evaluations that were needed on average in every outer iteration by each free vertex of Egypt mesh when Lo-D1-hS-SD-TC2 configuration was used. Note that vertices are sorted by element evaluations from largest to smallest. This figure shows that there is a large range of workloads per vertex (black line). Given that the optimization of a free vertex in the parallel algorithm is a serial process and using Equation 1 with constant α, this workload variability means that each vertex requires a runtime that can range in a large interval.

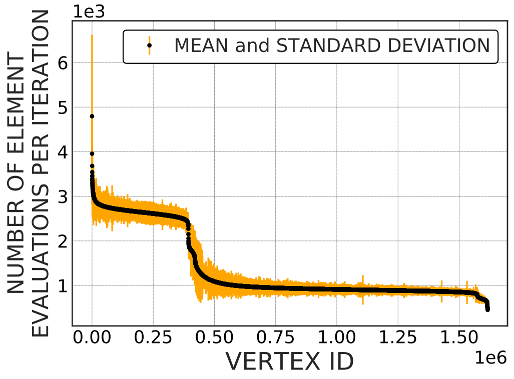
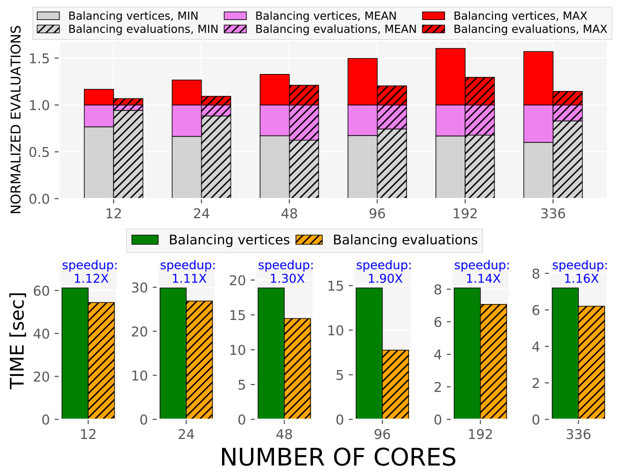


Fig. 5: Element evaluations per vertex and mesh iteration of a VrPA algorithm (configuration: Lo-D1-hS-SD-TC2) when it was employed to untangle and smooth the Egypt mesh using one compute node of *Cluster1* and 16 mesh partitions. Mesh vertices are sorted by element evaluations from largest to smallest.

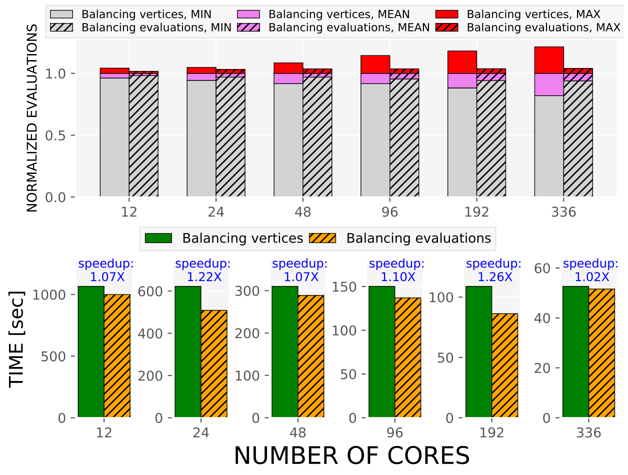
Equations 5 and 8 show that the main workload of a partition (P_i) is due to the element evaluations of all assigned vertices. Equations 6 and 9 show that load imbalance is caused by the difference in element evaluations between the most loaded partition and the average partition. Thus, we might expect that load balancing would improve when mesh partitioning uses the sum of workloads assigned to partitions rather than the sum of vertices.

8.2 Mesh partitioning by workload decomposition

Our hypothesis was tested in a new experiment by comparing the performance of parallel VrPA algorithms that use meshes partitioned both with



Mesh: Screwdriver. VrPA configuration: Lo-D1-hS-CG-TC2



Mesh: Egypt. VrPA configuration: Lo-D1-hS-SD-TC2

Fig. 6: Comparison of mesh partitioning strategies that balance either vertices or mesh element evaluations. MIN, MEAN and MAX denote the minimum, mean and maximum numbers of element evaluations per mesh partition, respectively. The element evaluations for each number of partitions were divided by the mean value. Speedup denotes the ratio of execution times.

and without workload information. The new proposal of mesh partitioning needs to know the workload of every vertex. This information cannot be obtained from the input mesh. Thus, a previous mesh iteration of the parallel optimization is used to derive the weight that represents the workload of a vertex. For this previous stage, the mesh is partitioned without workload information using `gpmetis` program. The second stage of our method consists in repartitioning the input mesh, including the weights.

8.3 Results

The reduction in load imbalance is significant as can be seen in Figure 6. This figure shows both the maximum and minimum numbers of element evaluations per mesh partition normalized to the mean number of evaluations.

Consequently, the execution times decreased in this parallel experiment. Our proposal achieved average speedups of 1.28X and 1.13X when Screwdriver and Egypt meshes were optimized, respectively. The extra times of both the previous mesh iteration and another mesh partitioning were added to the evaluation of our proposal.

Performance improvement is not as high for Egypt mesh as it is for Screwdriver mesh because the significance of load imbalance is smaller (see Figure 4). Thus, our mesh repartitioning strategy achieves larger performance improvement when the relevance of load imbalance is greater.

Since the main cost of our proposal is the previous optimization stage, larger benefits will be achieved when its execution time is much smaller than the total time to convergence. A circumstance where our proposal has a beneficial effect on performance occurs when the variance of the number of elements evaluations per vertex in successive mesh iterations is low (see Figure 5). In this case, a single mesh iteration is sufficient to derive the relative workloads of vertices that are valid for the rest of the iterations.

9 Conclusions

We have proposed a performance model for vertex repositioning algorithms on distributed-memory computers. This model is based on a workload measure called *number of mesh element evaluations*. The parallel model has been shown to be accurate with low average errors across a range of configurations in terms of the number of parallel processes, processor microarchitecture, mesh geometry, and algorithm configuration utilized. Furthermore, the parallel model was used to quantitatively understand the performance scalability, load balancing and synchronization and communication overheads. The results in this paper have shown that imbalance in the number of element evaluations and synchronization between boundary partitions are the major causes of the parallel bottlenecks. Finally, we have proposed a new approach to mesh partitioning that uses the number of mesh element evaluations to distribute vertices among parallel processes. This mesh partitioning strategy has been shown to reduce the imbalance in element evaluations caused by multilevel k-way partitioning algorithms. Consequently, our mesh repartitioning proposal improves the parallel performance of VrPA algorithms. As future work, we will study if performance improvement may be achieved using our strategy in other known load balancing techniques. Additionally, we will also investigate how to reduce the synchronization overhead of distributed-memory algorithms for repositioning mesh vertices.

Acknowledgement

This work has been supported by Spanish Government, "Secretaría de Estado de Universidades e Investigación", "Ministerio de Economía y Competitividad" and FEDER, grant contract: CTM2014-55014-C3-1-R. *Cluster2* (TeideHPC) was provided by the "Instituto Tecnológico y de Energías Renovables, S.A.". We thank to anonymous reviewers for their valuable comments and suggestions on this manuscript.

References

1. K. Barker, N. Chrisochoides: Practical Performance Model for Optimizing Dynamic Load Balancing of Adaptive Applications. In: Proc. 19^{th} IPDPS, 28.a-28.b, 2005.
2. D. Benitez, J.M. Escobar, R. Montenegro, E. Rodriguez: Performance Comparison and Workload Analysis of Mesh Untangling and Smoothing Algorithms. In: Proc. 27^{th} International Meshing Roundtable, 2018.
3. D. Bozdag, A. Gebremedhin, F. Manne, E. Boman, U. Catalyurek: A framework for scalable greedy coloring on distributed memory parallel computers. Journal of Parallel and Distributed Computing, 68(4):515-535, 2008.
4. M. Brewer, L. Diachin, P. Knupp, T. Leurent, D. Melander: The Mesquite Mesh Quality Improvement Toolkit. In. Proc. 12^{th} International Meshing Roundtable, 239-250, 2003.
5. N. Chrisochoides: A Survey of Parallel Mesh Generation Methods. Tech. Rep. SC-2005-09. Brown University, 2005.
6. L. Diachin, P. Knupp, T. Munson, S. Shontz: A Comparison of Inexact Newton and Coordinate Descent Mesh Optimization Techniques. In: Proc. 13^{th} International Meshing Roundtable, 243-254, 2004.
7. J.M. Escobar, E. Rodríguez, R. Montenegro, G. Montero, J.M. González-Yuste: Simultaneous untangling and smoothing of tetrahedral meshes. Comp.Meth.Appl.Mech.Eng., 192, 2775-2787, 2003.
8. L. Freitag, M.T. Jones, P.E. Plassmann: A parallel algorithm for mesh smoothing. SIAM J. Sci. Comput., 20(6):2023-2040, 1999.
9. C. Geuzaine and J.F. Remacle: Gmsh: A three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. Int. J. Numerical Methods in Engineering, 79(11):1309-1331, 2009.
10. W. Gropp, L. N. Olson, and P. Samfass: Modeling MPI Communication Performance on SMP Nodes. In: Proc. 23^{rd} European MPI Users Group Meeting, 2016.
11. G. Karypis: METIS (version 5.1.0) - A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-reducing Orderings of Sparse Matrices. Univ. of Minnesota, 2013.
12. P. Knupp: Updating meshes on deforming domains: An application of the target-matrix paradigm. Commun. Num. Method Eng., 24:467-476, 2007.
13. M. Mathis, D. Kerbyson: A general performance model of structured and unstructured mesh particle transport computations. J. Supercomputing, 34:181-199, 2005.
14. R. Montenegro, J.M. Cascón, J.M. Escobar, E. Rodríguez, G. Montero: An automatic strategy for adaptive tetrahedral mesh generation. Appl. Num. Math., 59(9):2203-2217, 2009.
15. T. Panitanarak, S.M. Shontz. A parallel log barrier-based mesh warping algorithm for distributed memory machines. Engineering with Computers, (34):59-76, 2018.
16. A. Sarje, S. Song, D. Jacobsen, K. Huck, J. Hollingsworth, A. Malony, S. Williams, L. Oliker: Parallel performance optimizations on unstructured mesh-based simulations. Procedia Computer Science, 51:2016-2025, 2015.
17. S.P. Sastry, S.M. Shontz: A parallel log-barrier method for mesh quality improvement and untangling. Engineering with Computers, 30(4):503-515, 2014.
18. S.P. Sastry, S.M. Shontz, S.A. Vavasis: A log-barrier method for mesh quality improvement and untangling. Engineering with Computers, 30(3):315-329, 2014.
19. R.B. Schnabel: Concurrent Function Evaluations in Local and Global Optimization. CU-CS-345-86. Comp. Science Tech. Rep. 332. Univ. Colorado, Boulder, 1986.