Performance model for mesh optimization on distributed-memory computers

D. Benitez, J.M. Escobar, R. Montenegro, E. Rodriguez
SIANI institute & DIS department, University of Las Palmas de Gran Canaria
Las Palmas de Gran Canaria, Spain

ABSTRACT

Many mesh optimization applications are based on vertex repositioning algorithms (VrPA). The execution times of these numerical algorithms vary widely, usually with a trade-off between different parameters. In this work, we analyze the impacts of six parameters of sequential VrPA on runtime. Our analysis is used to propose a new workload measure called *number of mesh element evaluations*. Since the execution time required for VrPA programs may be too large and there is concurrency in processing mesh elements, parallelism has been used to improve performance efficiently. The performance model is extended to parallel VrPA algorithms that are implemented in MPI. This model has been validated using two Open MPI versions on two distributed-memory computers and is the basis for the quantitative analysis of performance scalability, load balancing and synchronization and communication overheads. Finally, a new approach to mesh partitioning that improves load balancing is proposed.

KEYWORDS

Performance modeling, parallel numerical methods, parallel mesh optimization, load balancing.

ACM Reference Format:
D. Benitez, J.M. Escobar, R. Montenegro, E. Rodriguez. 2018. Performance model for mesh optimization on distributed-memory computers. In *25th European MPI Users' Group Meeting (EuroMPI '18), September 23–26, 2018, Barcelona, Spain.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3236367.3236372

1 INTRODUCTION

A mesh composed of vertices, edges and elements is used in numerical solvers of differential equations. 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 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]. Vertex repositioning algorithms (VrPA) have been adopted by a vast majority of mesh optimization applications [6–8, 17, 18], but no performance model for distributed-memory computers has been proposed yet.

Our contributions are: (1) a performance model for loosely synchronous VrPA programs executed on distributed-memory computers is proposed; (2) this parallel model is based on a new workload measure; (3) the performance scalability, load balancing and synchronization and communication overheads of VrPA algorithms is studied; (4) a new approach to mesh partitioning that reduces load imbalance is proposed.

The paper is organized as follows. Section 2 describes generalized versions of the sequential and parallel VrPA algorithms. We use experimental evidences to propose the new performance model. Thus, Section 3 describes the experimental setup and Sections 4 & 5 the sequential performance evaluation. After that, Sections 6 & 7 explain the parallel model. Section 8 analyzes the scalability and parallel overheads of VrPA algorithms. Section 9 describes the new load balancing proposal. Finally, the main conclusions are given.

2 GENERALIZED ALGORITHMS

VrPA algorithms improve the quality of a mesh by moving its free vertices. They can be posed as numerical optimization 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. 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, "Gl-D1-hS-SD-TC2".

2.1 Sequential algorithm

Many mesh optimization applications employ a VrPA that is similar to Algorithm 1 [6, 7, 18]. It consists of a variable

EuroMPI '18, September 23–26, 2018, Barcelona, Spain

ACM ISBN 978-1-4503-6492-8/18/09…$15.00
https://doi.org/10.1145/3236367.3236372

Table 1: Free VrPA parameters and their choices that are considered in this paper.

Parameter	Options		Related work
Objective function approach (A): $K = \sum_i^n f(q_i)$ n: total free elements*	**Gl**: All-vertex	$n = N_M$: free elements* of mesh	[6]
	Lo: Single-vertex	$n = N_v$: free elements* of local patch	[6]
Objective function formulation: $f(q_i)$ q_i: quality of i^{th} element $q_{min} = min(q_i)_{i\in\{1...n\}}$ $h(z) = \frac{1}{2}\left(z + \sqrt{z^2 + 4\delta^2}\right)$ δ, μ = 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	$f(q_i) = n^{-1}q_{min}^{-1} - \mu\ log(q_{min}^{-1} - q_i^{-1})$	[2]
	log2: Logarithmic barrier 2	$f(q_i) = n^{-1}q_{min} + \mu\ log(q_i - q_{min})$	[17]
	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 $h(z) = \frac{1}{2}\left(z + \sqrt{z^2 + 4\delta^2}\right)$ $\sigma_i = determinant(S_i)$ triangle: $d = 2, s = 3$ tetrahedron: $d = 3, s = 6$ a, b, δ, λ= constants	**hS**: Regularized mean ratio	$q_i = \frac{d\ [h(\sigma_i)]^{2/d}}{\|\|S_i\|\|_F^2}$	[7]
	MQ: Hybrid quality metric vol= element volume l_j= element side lengths	$q_i = \frac{\lambda\ vol}{1+e^{a\ \lambda\ vol}} + \frac{A}{1+e^{-b\ A}}$ $A = \frac{vol}{L^{d/2}}$ $L = \sum_{j=1}^{s} l_j^2$	[17]
	TU: Untangle quality metric	$q_i = 2\Big(-\sigma_i + \sqrt{\sigma_i^2 + \delta^2}\Big)^{-1}$	[4]
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 $Q_{min} = min(Q_i)_{i\in\{1...N_M\}}$	**TC1** (elements are not inverted)	$true$ = ($Q_{min} > 0$)	
	TC2 (output mesh is optimum) $\overline{Q}$: 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.

number of *mesh sweeps*. In each of them, every one of the free vertices is processed and can be repositioned if required by the numerical optimization method. The vertices that lie on the mesh surface are treated as fixed and are not updated.

The most time-consuming operation called $VertexReposi-tioning$ moves free vertices (V) of an input mesh (M). It iterates an *inner loop* while a minimum of the objective function (K) is being reached by using a numerical method (NM). K is constructed with A, f and q. $LogicFunction$ uses termination criteria (TC) to stop the algorithm. The *outer loop* is iterated in $Main$ procedure while $LogicFunction$ is not true. In each outer iteration, the spatial coordinates of all free vertices (X_V) are updated, and so a mesh sweep is implemented. $GlobalMeasures$ provides the average and minimum mean-ratio quality metric of the mesh [6]. At the end of the algorithm, an optimized mesh is obtained.

2.2 MPI algorithm

Algorithm 2 shows a generalized VrPA for distributed-memory computers that is similar to others [8, 17]. The input is a set of nC files, one for each mesh partition, P_i. 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 mesh is partitioned by assigning each vertex to one partition. In this way, the number of send and receive MPI 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

Algorithm 1 - Sequential mesh vertex repositioning algorithm.

```
1: ▷ Input: file with information of M mesh
2: #define: approach (A), formulation (f), quality metric (q), numerical minimization method (NM)
3: #define termination criteria: TC = LogicFunction(Q_min, ΔQ̄, ΔQ_min)
4: #define constants: τ = 10^-6 (maximum increase), N_maxInIter = 150 (maximum number of inner iterations), N_maxOutIter = 100 (maximum number of outer iterations)
5: N_e ← 0                                   ▷ Global variable: number of mesh element evaluations
6: procedure VertexRepositioning(W, X, n)
7: ▷ Inputs : W(free vertices), X(their coordinates), n(number of elements)
8:   ▷ Initiation : K = 0, ΔK = 0, m = 0 (inner loop index)
9:   while ΔK ≤ τ and m ≤ N_maxInIter do                          ▷ Inner loop
10:    X̂ ← X                                 ▷ Returned spatial coordinates (X̂) of vertices (W)
11:    ▷ Initiation : P ← 0                  ▷ Moving directions: P = {p_v}, v ∈ W
12:    for i = 1, ..., n do                  ▷ n: number of free elements
13:      for each free vertex v of i^th free element do
14:        p_v += NM(f'(q_i), v)             ▷ f': derivatives used in NM
15:      N_e += 1                            ▷ Number of mesh element evaluations
16:    X ← X̂ + P                             ▷ Tentative positions of free vertices
17:    K_t ← 0                               ▷ Initial value of objective function
18:    for i = 1, ..., n do
19:      K_t += f(q_i)                       ▷ MESH ELEMENT EVALUATION
20:      N_e += 1                            ▷ Number of mesh element evaluations
21:    ΔK ← K_t − K
22:    K ← K_t                               ▷ Final value of objective function
23:    m += 1                                ▷ Number of inner iterations
24:  return X̂                                ▷ Output: updated coordinates of free vertices
25: procedure main( )
26:  ▷ Read the vertex and element information of M mesh
27:  Q_min ← GlobalMeasures(M)               ▷ Minimum quality of input mesh
28:  ▷ Initiation : ΔQ̄ = 10^6, ΔQ_min = 10^6, k = 0 (loop index)
29:  while TC ≠ true and k ≤ N_maxOutIter do                      ▷ Outer loop
30:    if A = Gl then                        ▷ Gl : all-vertex approach
31:      X_V ← VertexRepositioning(V, X_V, N_M)
32:    else                                  ▷ Lo : single-vertex approach
33:      for each free vertex v ∈ M do
34:        x_v ← VertexRepositioning(v, x_v, N_v)
35:    (Q_min, ΔQ̄, ΔQ_min) ← GlobalMeasures(M)
36:    k += 1                                ▷ Number of mesh/outer iterations
37:  ▷ Output: file with information of optimized M mesh
```

Algorithm 2 - Parallel mesh vertex repositioning algorithm.

```
1: ▷ Input: files with information of P_i partitions, P_i ← Partition(M)
2: ▷ Defines and subroutines as in Algorithm 1
3: N_{e,i} ← 0    ▷ element evaluations for partitions P_i, i ∈ {1, ..., nC}
4: procedure MAIN( )
5:     for P_i ∈ M in parallel do                    ▷ Parallel phase 1: begin
6:         ▷ Read the vertex and element information of P_i partition
7:         I_i ← BOUNDARYCOLORING(P_i)
8:                                   ▷ I_i = {I_ij}_{j∈{1..nF_i}, i∈{1..nC}}
9:         MPI_Send-MPI_Receive information of boundary/ghost vertices
10:        ▷ Store the order of partition free boundary/ghost vertices
11:        Q_{min,i} ← GLOBALMEASURES(P_i)    ▷ Initial partition quality
12:    Synchronization MPI_Allreduce              ▷ Parallel phase 1: end
13:    for P_i ∈ M in parallel do
14:        ▷ Initiation : ΔQ̄_i = 10^6, ΔQ_{min,i} = 10^6, k_i = 0 (loop index)
15:        while TC ≠ true and k_i ≤ N_{maxOutIter} do          ▷ Outer loop
16:            if A = Gl then     ▷ Par. pha. 2 - Interior processing: begin
17:                X_V ← VERTEXREPOSITIONING(V, X_V, N_M)
18:            else                                  ▷ A = Lo (single-vertex)
19:                for each free interior vertex v ∈ P_i do
20:                    x_v ← VERTEXREPOSITIONING(v, x_v, N_v)
21:            Synchronization MPI_Barrier          ▷ Parallel phase 2: end
22:                                                 ▷ Parallel phase 3: begin
23:            for each boundary independent-set I_ij ∈ P_i do
24:                for each free boundary vertex of partition v ∈ I_ij do
25:                    x_v ← VERTEXREPOSITIONING(v, x_v, N_v)
26:                MPI_Send-MPI_Receive coordinates of vertices x_v
27:            Synchronization MPI_Barrier          ▷ All boundary vertices
28:            (Q_{min,i}, ΔQ̄_i, ΔQ_{min,i}) ← GLOBALMEASURES(P_i)
29:            k_i + = 1
30:            Synchronization MPI_Allreduce        ▷ Parallel phase 3: end
31:        MPI_Send-MPI_Receive N_{e,i}
32:    ▷ Output: files with information of optimized P_i partitions
```

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.

Each partition is assigned to a different MPI 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 5 to 12. 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 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 7) [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 9).

Finally, a list with the order of boundary and ghost vertices is created in line 10. This list determines the order in which these vertices are optimized in the MPI process or received from other MPI processes in phase 3. Interior vertices do not need to be reordered because all adjacent vertices are assigned to the same MPI process. Using the function `MPI_Allreduce` at the end of phase 1, a synchronization barrier ensures that all partitions have completed these steps before continuing computation (line 12).

In each mesh sweep, also called *outer* or *mesh iteration*, all interior and boundary vertices are optimized separately in parallel phases 2 (lines 16...21) and 3 (lines 22...30), respectively, adjusting the spatial coordinates x_v of each free vertex v. The optimization method is the same as Algorithm 1 (lines 17, 20, 25). Thus, the parallel performance is based on the performance of the underlying serial numerical method.

Interior vertices of every partition are not dependent on vertices of other partitions and are sequentially optimized by the same MPI process. In this way, the interior vertices of all partitions are optimized in parallel. A single synchronization phase among partitions is established to ensure that all interior vertices are completely repositioned (line 21).

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 send/receive MPI functions (line 26). These computation-synchronization-communication phases are repeated until all boundary vertices have been optimized (lines 23...26).

When all boundary vertices have been updated (line 27), the minimum quality metrics of all partitions are calculated and distributed (lines 28...30) and the mesh sweep finishes. At this moment, a new mesh sweep may begin if convergence conditions are not met (line 15). After a variable number of mesh sweeps, the output of our parallel algorithm provides optimized mesh partitions (line 32).

3 EXPERIMENTAL SETUP

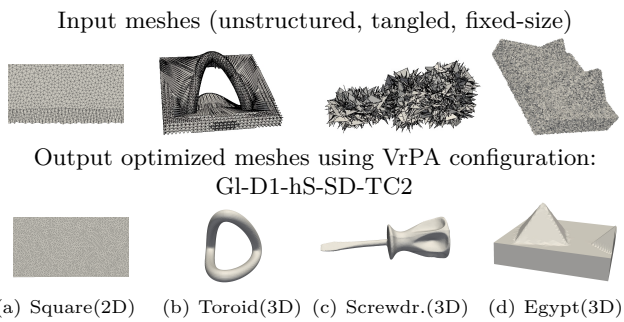
We developed programs that include double-precision floating-point data structures and functions from *MPI* and the *Mesquite* C++ library [4], which is specialized in mesh smoothing. *Mesquite* was extended to support *hS* and *MQ* quality metrics, *log1*, *log2* and *inv* objective function formulations and *TC2* termination criterion (see Table 1). We used *Open MPI* (3.1.0, 1.6.5), and *gcc* (4.4.7, 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%.

Algorithms 1 and 2 were applied to the 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].

Numerical experiments were conducted on two cluster computers called *Cluster1* and *Cluster2* that are in two different

Table 2: Characteristics of the 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 ($\overline{Q}$)	0.953	0.171	0.130	0.230
Standard deviation of the mean-ratio metric (σ_Q)	0.047	0.312	0.214	0.268



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

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

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 Westmere-EP cores each, 2.4 GHz), and 48 GB of DDR3/1333 MHz ECC RAM. 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 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 Sandy Bridge-EP E5-2670@2.6GHz cores and 32 GB of DDR3/1600 ECC RAM 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

The first performance model proposed in this work for sequential VrPA algorithms tries to justify their execution times. Using *Cluster1*, Figure 2 shows the execution times of 68 sequential algorithms for mesh untangling. Except for the TC1 convergence criterion, the input mesh and rest of VrPA parameters are free. Note that not all parameter combinations are depicted. It is due to the existence of inverted elements in the output meshes; i.e., TC1 is not met.

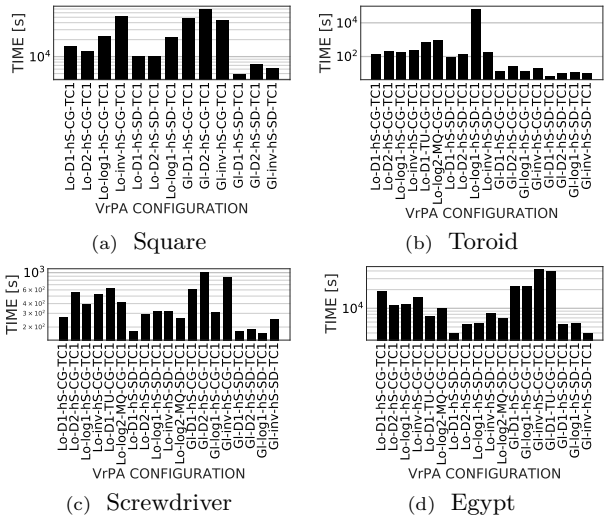
Some authors have shown that the execution time of VrPA algorithms is directly proportional to mesh size [18]. This is true only when the VrPA configuration is fixed and the total numbers of inner and outer iterations are both fixed. Figure 3(a) shows a graph of mesh size versus time that was obtained from the above-mentioned configurations that successfully untangle a mesh. Note that 11 symbols are used to represent 68 performance results. The same symbol represents the results derived from configurations that employed different solvers ($\langle NM \rangle$={CG, SD}), were applied to distinct meshes and fixed the rest of the parameters. Taking the 68 configurations, the correlation between time and mesh size taken in the linear scale is r=0.43. Fixing the TC1 convergence criterion does not limit generality as results for TC2 have similar correlation coefficient.

4.1 The number of mesh element evaluations

N_e in Algorithm 1 is called *number of mesh element evaluations* and takes into account multiple evaluations of an element quality metric and its derivative. This measure involves computing the separable but not independent parts of objective function evaluations. Although not exactly the same definition, mesh element evaluation is similar to the *concurrent function evaluation step* defined in [19] for identifying parallelism opportunities in finite difference gradients.

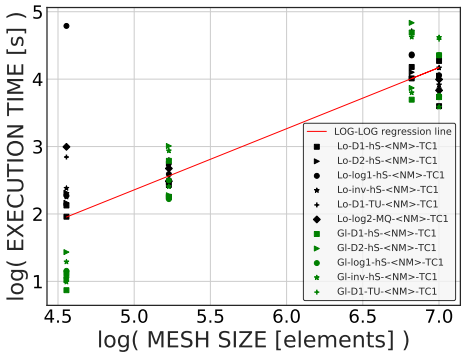
Figure 3(b) shows a graph of N_e versus time for the above mentioned 68 VrPA configurations. In this case, the correlation coefficient between time and N_e taken in the linear scale is r=0.94. Therefore, execution time is more directly proportional to mesh element evaluations than mesh size.

It is important to note that N_e is not very intrusive and depends not only on the problem size but also on the number of inner and outer iterations required to meet the convergence criteria. If N_e was known before computation, the execution time of a VrPA algorithm could be predicted. Estimating N_e a priori is a difficult and open problem since it depends in a

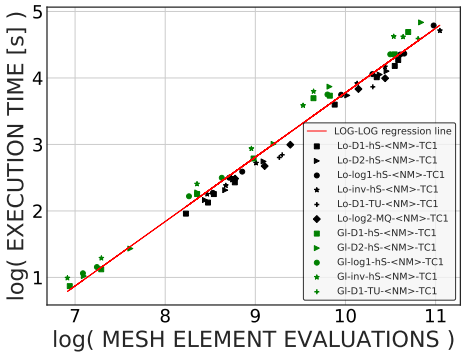


(a) Square (b) Toroid (c) Screwdriver (d) Egypt

Figure 2: Performance of mesh untangling algorithms.



(a) Mesh size vs. time, r=0.43



(b) Element evaluations vs. time, r=0.94

Figure 3: Scalability of mesh untangling algorithms (TC1).

non-deterministic way on all VrPA parameters considered in this work. However, this measure can be used in practice to quantitatively justify real performance and fine-tune parallel programs. We will use the number of element evaluations as workload measure in two new performance models.

5 SEQUENTIAL PERFORMANCE MODEL

Taking the findings of the previous section, we use a simple one-parameter model to understand the performance of sequential VrPA algorithms,

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

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. In this way, the time to optimize a mesh is directly proportional to the number of element evaluations.

This model may justify previous experimental observations where more element evaluations cause usually larger runtimes. However, there are VrPA configurations with fewer element evaluations than others that require more runtime (see Figure 3(b)). This is due to the fact that each VrPA configuration causes a different time per element evaluation.

5.1 Model application and accuracy

To check the accuracy of this model, we used the 68 VrPA configurations that met the TC1 convergence criterion. Randomly chosen, half of the configurations were used to obtain α.

Table 3: Relative errors of the sequential performance model.

Free VrPA parameter	CPU: E5645		CPU: E5-2670	
	Mean	Max	Mean	Max
Approach (A)	0.28	0.61	0.19	0.61
Formulation (f)	0.07	0.12	0.06	0.17
Quality metric (q)	0.05	0.10	0.05	0.11
Numerical method (NM)	0.08	0.17	0.08	0.16
Convergence criteria (TC)	0.01	0.02	0.01	0.21
Mesh	0.07	0.20	0.09	0.41

We calculated for each configuration the ratio of time to element evaluations: $\alpha = t_{CPU}^{Sreal}/N_e$. After averaging the results, $\overline{\alpha}$ was 6.7 10^{-7} and 4.9 10^{-7} [sec/elem] when E5645 and E5-2670 CPUs were used, respectively. The relative errors of estimates were obtained with the other half of configurations. The average errors were 0.27 (E5645) and 0.34 (E5-2670).

We extended this accuracy analysis by setting free only one of the five VrPA parameters (see Table 1) or the input mesh. For this experiment, we analyzed a total of 136 VrPA configurations that met the convergence criteria TC1 or TC2.

Taking groups of configurations that have one free and five fixed VrPA parameters, we obtained a mean time per element evaluation ($\overline{\alpha}$) in each group. Then, we compared the time provided by our model (Eq.1) with the real execution time (t_{CPU}^{Sreal}) of every configuration in each group. For each free parameter, the mean and maximum relative errors using the above-mentioned CPUs are shown in Table 3.

The errors are caused by the variability in α of the VrPA configurations that constitute each group. The largest variability about the mean occurs when the approach parameter is free. Our model fits best when it is applied to a specific VrPA algorithm and mesh. Moreover, our results indicate that the model parameter (α) depends on the processor architecture. This one-parameter model is the basis of the performance model for parallel VrPA algorithms that is described below.

6 PARALLEL PERFORMANCE MODEL

In this section, we describe a new performance model to justify the parallel runtimes of Algorithm 2 for a selected VrPA configuration on a determined distributed-memory computer. This model uses the time per mesh element evaluation (α) that is obtained from the sequential execution of the same configuration using Algorithm 1. Then, selecting one VrPA configuration and using Equation 1,

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

We assume that this model parameter is constant for all parallel experiments that use the same VrPA configuration and cluster computer. Since there is an MPI barrier between the repositioning of interior and partition boundary vertices, 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,

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

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 of two components,

$$t^{Pmodel}_{interior} = t^{Pmodel}_{scalable,interior} + t^{Pmodel}_{imbalance,interior} \quad (4)$$

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

$$t^{Pmodel}_{scalable,interior} = \alpha \frac{N^{Pmodel}_{e,interior}}{nC} \quad (5)$$

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 $N^{Pmodel}_{e,interior,max}$ is the maximum number of interior element evaluations of a partition.

$$t^{Pmodel}_{imbalance,interior} = \alpha\left(N^{Pmodel}_{e,interior,max} - \frac{N^{Pmodel}_{e,interior}}{nC}\right) \quad (6)$$

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,

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

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

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

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,

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

where $N^{Pmodel}_{e,boundary,j,max}$ is the maximum number of element evaluations of the j 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 (Eq. 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 the 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.

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

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.

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

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 the 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. For each vertex, we use a 32-byte block to send/receive spatial coordinates and global ID. γ denotes the average number of partitions that share the same vertex, and $nV_{boundary}$ the number of free boundary vertices of all partitions. LAT was determined by performing an MPI latency timing test in both clusters: $LAT = 2\ 10^{-6} sec$. BW was obtained using code instrumentation and performing an MPI bandwidth test. At first, we measured the average size of MPI messages for each VrPA configuration and number of partitions. Then, BW was assigned the average bandwidth provided by the MPI test for each message size. In our experiments, BW ranged from 0.36 to 18.0 Gbit/sec in both clusters. The rest of the parameters, β, γ and $nV_{boundary}$, are obtained from the partitions of the input mesh at the end of the partitioning phase.

7 VALIDATION OF THE PARALLEL MODEL

The point of this section is to validate the parallel model using different VrPA parameters, meshes and number of MPI processes. Our experiments involved a total of 136 configurations, half of them met the TC1 convergence criterion, and the other half met TC2. Due to lack of space and the number of parallel results, we have selected four configurations. Since the performance of sequential algorithms was explained using mesh untangling problems (TC1), the selected configurations

simultaneously untangle and smooth meshes (TC2). The rest of parameters cover most of the choices shown in Table 1.

Figures 4 and 5 show 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 ($Cluster2$).

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. In the next section, the parallel model is employed to study the performance scalability, load balancing and overheads of VrPA algorithms.

8 MPI PERFORMANCE ANALYSIS

Figure 6 shows stacked column graphs for the times provided by our parallel model when $Cluster1$ was used to run Algorithm 2. Every single column is divided into six sections that are grouped into four categories called *scalable parallelism*, *imbalance*, *synchronization* and *communication*. Next, these results are analyzed and discussed. Due to lack of space in this paper, the analysis of categories only considers results obtained from $Cluster1$.

Scalable parallelism includes runtimes for optimizing interior and boundary vertices if the sequential workloads were evenly distributed over all MPI processes (Eq. 5 & 8). These times are represented in Figure 6 by the two bottom columns denoted as *Inter-Scaling* and *Boun-Scaling*, respectively.

As the number of processes increases in strong scaling when a mesh is optimized, we can observe that the time devoted to this category reduces. It is due to the fact that we are solving fixed-size problems and the element evaluations in each partition reduce. Note in Figure 6 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.

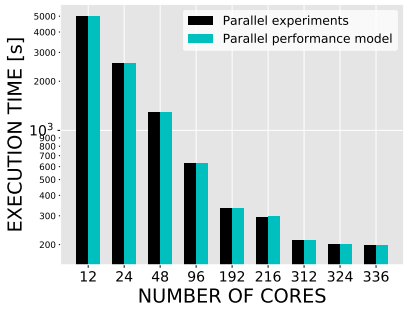
Another increasing trend is observed when problems of different sizes are compared for a given number of cores. For example, using 324 cores in $Cluster1$, note that the fraction of time in scalable parallelism is 25% for Screwdriver mesh, 68% for Egypt mesh and 78% for Square mesh. Since for 324 cores 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 (Eq. 6 & 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 Fig. 6). 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 bottleneck. For example, using 324 cores in $Cluster1$, load imbalance is responsible for 11% and 19% of the total runtime, respectively.

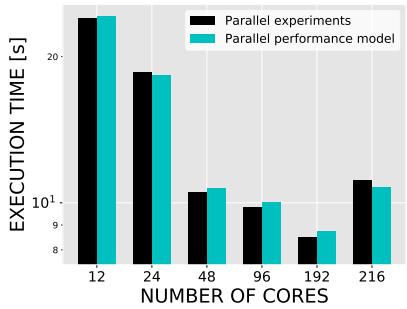
Synchronization (Eq. 10) includes the overheads caused by the independent sets of partition boundary vertices that have to be processed in the order determined in parallel phase 1. Note in Figure 6 that, as the number of MPI processes (nC) increases, the percentage relevance of this category tends to 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 MPI processes 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.

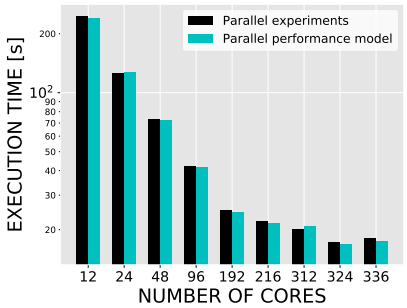
Communication is a category that considers the overhead caused by the transmission of updated coordinates of partition boundary vertices (Eq. 11). This overhead increases with the number of partitions because it depends on the numbers of boundary vertices and independent sets. However, its percentage relevance is the lowest, from 1% to 2% when 336 cores are used (see Fig. 6). Therefore, VrPA algorithms do not suffer significantly from the MPI communication overhead in our experiments. This is due to the dependence of



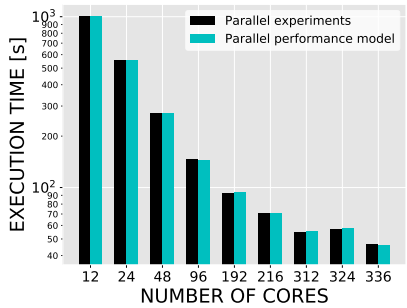
(a) **Square**
VrPA configuration: Lo-D2-hS-SD-TC2
serial: $t_{CPU}^{Sreal} = 5.8\ 10^4 sec,\ \alpha = 4.6\ 10^{-7} sec/elem$
output meshes: $\overline{Q_{min}} = 0.633 \pm 0.001$



(b) **Toroid**
VrPA configuration: Gl-inv-hS-SD-TC2
serial: $t_{CPU}^{Sreal} = 19.5\ sec,\ \alpha = 1.1\ 10^{-6} sec/elem$
output meshes: $\overline{Q_{min}} = 0.333 \pm 0.094$

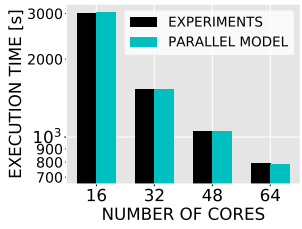


(c) **Screwdriver**
VrPA configuration: Gl-log1-hS-CG-TC2
serial: $t_{CPU}^{Sreal} = 1.6\ 10^3 sec,\ \alpha = 7.5\ 10^{-7} sec/elem$
output meshes: $\overline{Q_{min}} = 0.255 \pm 0.002$

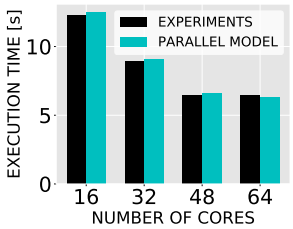


(d) **Egypt**
VrPA configuration: Lo-D1-hS-SD-TC2
serial: $t_{CPU}^{Sreal} = 1.1\ 10^4 sec, \alpha = 5.2\ 10^{-7} sec/elem$
output meshes: $\overline{Q_{min}} = 0.201 \pm 0.002$

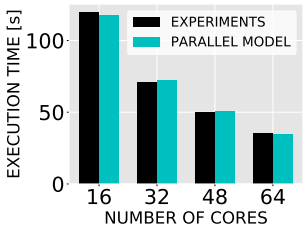
Figure 4: Results of the parallel experiments (t_{CPU}^{Preal}) and performance model (t_{CPU}^{Pmodel}) using *Cluster1* and *OpenMPI 1.6.5*.



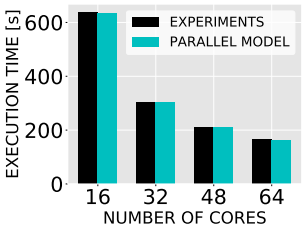
(a) **Square**
Lo-D2-hS-SD-TC2
$(t_{CPU}^{Sreal}, \alpha)$=$(4.6\ 10^4 sec, 0.4\ \mu sec/elem)$
$\overline{Q_{min}} = 0.633 \pm 0.001$



(b) **Toroid**
Gl-inv-hS-SD-TC2
$(10\ sec, 0.6\ \mu sec/elem)$
$\overline{Q_{min}} = 0.318 \pm 0.108$



(c) **Screwdriver**
Gl-log1-hS-CG-TC2
$(9.9\ 10^2 sec, 0.4\ \mu sec/elem)$
$\overline{Q_{min}} = 0.256 \pm 0.001$



(d) **Egypt**
Lo-D1-hS-SD-TC2
$(8.2\ 10^3 sec, 0.4\ \mu sec/elem)$
$\overline{Q_{min}} = 0.202 \pm 0.002$

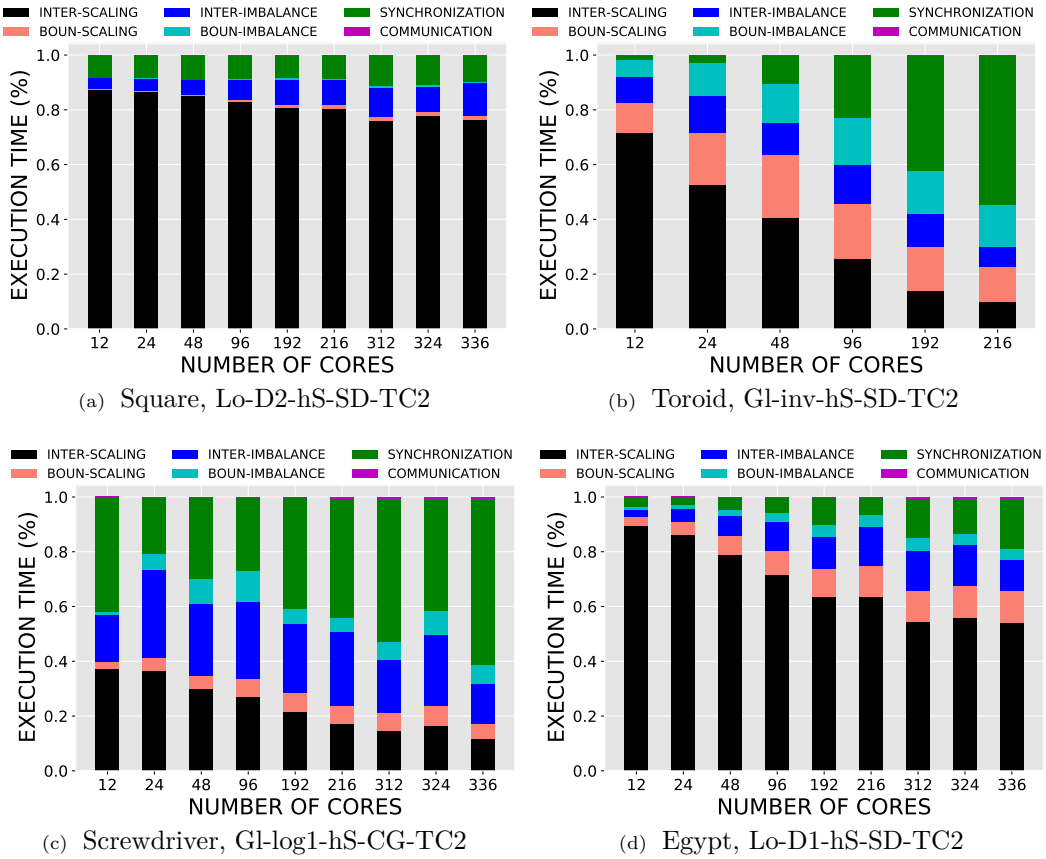
Figure 5: Results of the parallel experiments (t_{CPU}^{Preal}) and performance model (t_{CPU}^{Pmodel}) using *Cluster2* and *OpenMPI 3.1.0*.

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.

9 APPLICATION TO LOAD BALANCING

Parallel mesh optimization algorithms for distributed-memory computers use a previous phase of mesh partitioning to balance and distribute vertices among MPI processes [17]. As stated above, we used the Metis package that provides programs based on the multilevel graph partitioning paradigm [11]. These programs require as input a file storing a mesh. Part of this file contains information relevant for vertices.

The results of the previous section show that load imbalance is a significant overhead. To reduce this overhead, we propose to include in the input file 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



(a) Square, Lo-D2-hS-SD-TC2

(b) Toroid, Gl-inv-hS-SD-TC2

(c) Screwdriver, Gl-log1-hS-CG-TC2

(d) Egypt, Lo-D1-hS-SD-TC2

Figure 6: Time breakdowns provided by the parallel model when *Cluster*1 and *OpenMPI 1.6.5* were used.

vertices. With our proposal, this program balances element evaluations, e.g., the sum of evaluations of the vertices assigned to each MPI process is approximately the same across the partitions.

Figure 7 shows the element evaluations that were needed on average in every outer iteration by each free vertex of two meshes. 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). Using Equation 1 and assuming constant α, this means that each vertex requires a runtime that can range in a large interval.

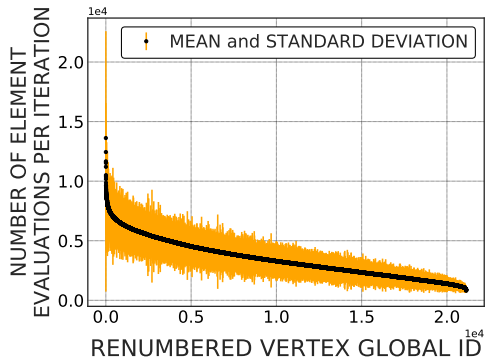
Equations 5 and 8 show that the main workload of a partition or MPI process (P_i) is due to the element evaluations of all assigned vertices. Equations 6 and 9 show that load imbalance is caused by the difference in element evaluations between the most loaded partition and the average partition. Thus, 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.

Our hypothesis was examined in a new experiment by comparing the performance of parallel VrPA algorithms that use meshes partitioned both with and without workload information. The reduction in load imbalance is significant as can be seen in Figure 8. This figure shows the maximum and minimum numbers of element evaluations per MPI process normalized to the mean number of evaluations.

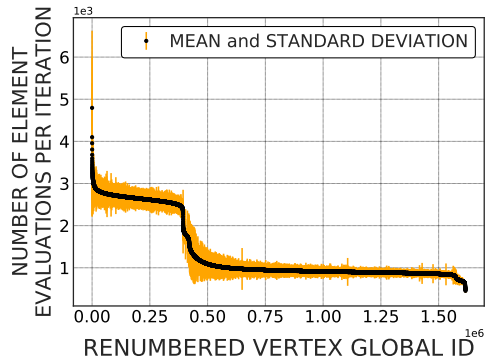
Consequently, the execution times were reduced in this parallel experiment. Our proposal achieved an average speedup of 1.28X and 1.13X when Screwdriver and Egypt meshes were optimized, respectively. The extra times of both the previous mesh iteration and another mesh partitioning were added to the evaluation of our proposal. Performance improvement is not as high for Egypt as it is for Screwdriver because the relevance of load imbalance is smaller (see Fig. 6).

10 CONCLUSIONS

We have proposed a performance model for parallel vertex repositioning algorithms on distributed-memory computers. This model involves a new 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 MPI processes, CPU microarchitecture, mesh geometry, and algorithm configuration utilized. Further, the parallel model was used to quantitatively understand the performance scalability, load balancing and synchronization and communication overheads. Finally, we have proposed a new approach to mesh partitioning that uses the number of mesh element evaluations to distribute vertices among MPI processes. This proposal reduces load imbalance and improves performance.

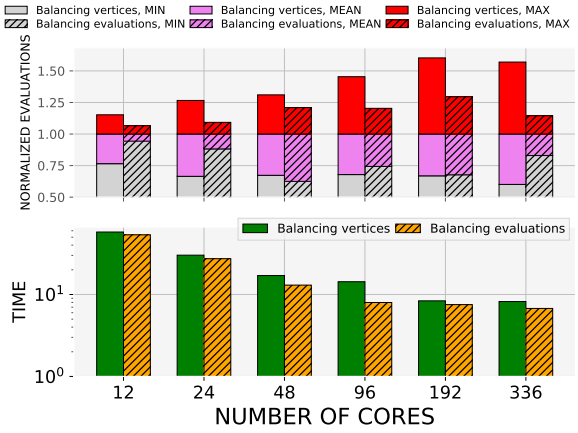


(a) Mesh: Screwdriver, VrPA: Lo-D1-hS-CG-TC2

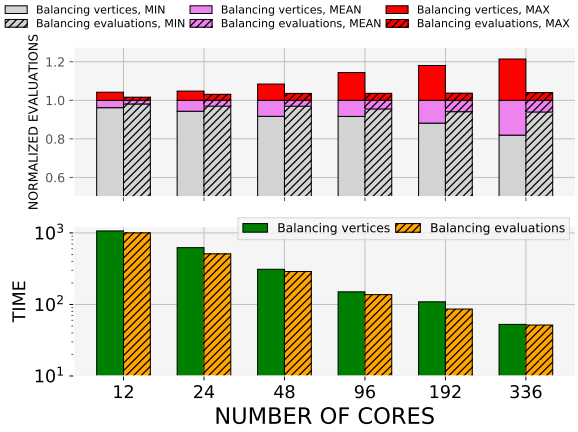


(b) Mesh: Egypt, VrPA: Lo-D1-hS-SD-TC2

Figure 7: Element evaluations per vertex and mesh iteration.



(a) Mesh: Screwdriver, VrPA: Lo-D1-hS-CG-TC2



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

Figure 8: Comparison of mesh partitioning strategies.

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. 19^{th} IPDPS, pp.28.a-28.b, 2005.
[2] M.S. Bazaraa, H.D. Sherali, C.M. Shetty: Nonlinear Programming: Theory and Algorithms, 3^{rd} Edition. Wiley, 2006.
[3] D. Bozdag, A. Gebremedhin, F. Manne, E. Boman, U. Catalyurek: A framework for scalable greedy coloring on distributed memory parallel computers. J. Par. Distrib. Comp. 68(4):515-535, 2008.
[4] M. Brewer, L. Diachin, P. Knupp, T. Leurent, D. Melander: The Mesquite mesh quality improvement toolkit. 12^{th} Int. Meshing Roundtable, pp.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. 13^{th} Int. Meshing Roundtable, pp.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, J.F. Remacle: Gmsh, a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. Int. J. Num. Meth. Eng. 79(11), pp.1309-1331, 2009.
[10] W. Gropp, L. N. Olson, and P. Samfass: Modeling MPI Communication Performance on SMP Nodes. Proceedings of the 23^{rd} European MPI Users Group Meeting, pp.41-50, 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 V. 51, pp.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.