Performance Comparison and Workload Analysis of Mesh Untangling and Smoothing Algorithms

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

Abstract This paper compares methods for simultaneous mesh untangling and quality improvement that are based on repositioning the vertices. The execution times of these algorithms vary widely, usually with a trade-off between different parameters. Thus, computer performance and workloads are used to make comparisons. A range of algorithms in terms of quality metric, approach and formulation of the objective function, and optimization solver are considered. Among them, two new objective function formulations are proposed. Triangle and tetrahedral meshes and three processors architectures are also used in this study. We found that the execution time of vertex repositioning algorithms is more directly proportional to a new workload measure called *mesh element evaluations* than other workload measures such as mesh size or objective function evaluations. The comparisons are employed to propose a performance model for sequential algorithms. Using this model, the workload required by each mesh vertex is studied. Finally, the effects of processor architecture on performance are also analyzed.

1 Introduction

Mesh optimizing techniques reduce the total time to solution and improve the accuracy of results of PDE solvers. Processing a mesh can spend up to 25% of the overall running time of a PDE-based application [5]. When a mesh element is inverted, standard finite element simulation algorithms cannot numerically solve the PDE, although some methods are being investigated to solve a PDE on a tangled mesh [20]. Thus, researchers and practitioners recommend untangling the mesh prior to analysis using commercial packages.

Vertex repositioning algorithms (VrPA) have been adopted by a vast majority of mesh optimization applications [6, 7, 17], including hex meshes [11, 14]. VrPA algorithms improve the quality of a mesh by moving its free vertices. 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). The execution times of these numerical algorithms vary widely, usually with a trade-off between parameters.

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

There are few studies that compare the performance of mesh optimization algorithms. Diachin et al. compared the performance of two VrPA algorithms [6]. One of them employed an inexact Newton method and the all-vertex approach to numerically optimize the global objective function. The other algorithm used a coordinate descent method and the single-vertex approach.

Sastry and Shontz compared the performance of several optimization methods for mesh quality improvement [15]. They considered all-vertex and single-vertex approaches in combination with gradient and Hessian-based optimization solvers. In their paper, it is showed that performance can vary significantly, depending on the choice of mesh quality metric.

These previous studies considered valid input meshes, smoothing algorithms, one objective function formulation, and one processor architecture. In our work, up to 34 algorithms that simultaneously untangle and smooth meshes composed of two different element topologies and three processor architectures are considered. Additionally, three mesh quality metrics and five objective function formulations, two of them new, are investigated (Section 2). A single generalized version of the sequential VrPA algorithm is used (Section 3), and its implementation details are explained in Section 4.

One of our goals is to determine when one of these methods for mesh untangling and smoothing and why one processor is preferable to the others (Section 5). Preference includes the execution time, success in untangling meshes and the quality of the optimum mesh produced. To gain insights into the causes of performance variability, the workload of VrPA algorithms is analyzed (Sections 6 & 7). Another goal is to model the performance of sequential VrPA algorithms. Thus, we propose a performance model for mesh optimization and its accuracy is studied in Section 8. We have also investigated the causes of the variability in time of VrPA methods (Section 9).

2 Free parameters of the study

The vertex repositioning problem has been formulated by other authors [6]. 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-D1-hS-SD-TC2".

2.1 Novel objective function formulations

Table 1 includes two new objective functions for mesh untangling and quality improvement called *Logarithmic distortion-based barrier* (Equation 1, **log1**) and *Regularized distortion-based barrier* (Equation 2, **inv**), where q_i is the quality metric function, n is the number of free elements involved in forming the objective function and μ, δ and τ are constants.

$$K = \frac{1}{q_{min}} - \mu \sum_{i=1}^{n} log(\frac{\tau}{q_{min}} - \frac{1}{q_i}) \qquad q_{min} = min(q_i)_{i \in \{1...n\}} \tag{1}$$

$$K = \sum_{i=1}^{n} \frac{1}{q_i} + \frac{1}{h\left(\frac{1}{q_{min}} - \frac{1}{q_i}\right)} \qquad h(z) = \frac{1}{2}\left(z + \sqrt{z^2 + 4\delta^2}\right) \tag{2}$$

These barrier objective functions are defined for differentiable quality metrics whose maximum and minimum values are the qualities of ideal and degenerate elements, respectively. This formulation assumes that the quality of an element should be maximized in order to obtain the ideal element. Thus, our new barrier functions are used in vertex repositioning methods that minimize the objective function.

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 $q_{min} = min(q_i)_{i \in \{1 \ldots n\}}$ $h(z) = \frac{1}{2}\left(z + \sqrt{z^2 + 4\delta^2}\right)$ $\delta, \mu, \tau =$ constants	**D1**: Distortion 1	$f(q_i) = q_i^{-1}$	[3]
	D2: Distortion 2	$f(q_i) = q_i^{-2}$	[3]
	log1: Logarithmic barrier 1	$f(q_i) = n^{-1} q_{min}^{-1} - \mu\ log(\tau\ q_{min}^{-1} - q_i^{-1})$	new, see 2.1
	log2: Logarithmic barrier 2	$f(q_i) = n^{-1} q_{min} + \mu\ log(q_i - q_{min})$	[16]
	inv: Regularized barrier	$f(q_i) = q_i^{-1} + \frac{1}{h\left(q_{min}^{-1} - q_i^{-1}\right)}$	new, see 2.1
Element quality metric: q_i S_i: Jacobian matrix $\|\|\ \|\|_F$: Frobenius norm $h(z) = \frac{1}{2}\left(z + \sqrt{z^2 + 4\delta^2}\right)$ $\sigma_i = determinant(S_i)$ $triangle : d = 2, s = 3$ $tetrahedron : d = 3, s = 6$ $a, b, \delta, \lambda = constants$	**hS**: Regularized mean-ratio	$q_i = \frac{d\ [h(\sigma_i)]^{2/d}}{\left\|\left|S_i\right|\right\|_F^2}$	[7]
	MQ: Hybrid quality metric vol = element volume l_j= element edge lengths	$q_i = \frac{\lambda\ vol}{1+e^{a\ \lambda\ vol}} + \frac{A}{1+e^{-b\ A}}$ $A = \frac{vol}{L^{d/2}} \qquad L = \sum_{j=1}^{s} l_j^2$	[16]
	TU: Untangle quality metric	$q_i = 2\left(-\sigma_i + \sqrt{\sigma_i^2 + \delta^2}\right)^{-1}$	[3]
Numerical minimization method (NM)	**CG**: Conjugate Gradient	Polack-Ribiere, analytical derivatives	[3]
	SD: Steepest Descent	Analytical derivatives	[3]
Termination criteria (TC) Q_i: mean-ratio quality value of the i^{th} element $Q_{min} = min(Q_i)_{i \in \{1 \ldots N_M\}}$	**TC1** (untangled mesh)	$true = (\ Q_{min} > 0\ )$	[3]
	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.

2.2 Calculation of the constant value for δ

Our experience has shown that the constant values involved in the calculation of quality metrics can determine whether a mesh untangling algorithm is successful in producing meshes without inverted elements. δ in Equation 2 and Table 1 represents a constant value that depends on the mesh in all-vertex approach ($n = N_M$) or submesh in single-vertex approach ($n = N_v$):

$$\delta = \max\{\ 10^{-3}\ \bar{\sigma},\ \mathrm{Re}(\ 10\ \sqrt{\epsilon(\epsilon - \sigma_{min})}\ )\ \} \tag{3}$$

$$\bar{\sigma} = \frac{1}{n}\ \sum_{i=1}^{n} |\sigma_i| \qquad \sigma_{min} = \min\{\sigma_i\}_{i\in\{1,\ldots,n\}} \qquad \epsilon = 10^{11} DBL\_EPS$$

where σ_i is defined in Table 1 and DBL_EPS is upper bound on the relative error due to rounding in floating-point arithmetic. δ is always recalculated before a different vertex is optimized by a single-vertex method. For all-vertex methods, δ is recalculated before a different mesh iteration begins.

This calculation of δ was applied to the known metric called *untangle quality metric* and denoted **TU** [3] (see Table 1). For the **TU** metric and the regularized mean-ratio quality metric (**hS**), this strategy is key to successfully producing a mesh without inverted elements when the input mesh is tangled.

3 Sequential VrPA algorithm

Many mesh optimization applications employ a VrPA that is similar to Algorithm 1 [6, 7, 17]. It consists of a variable number of mesh sweeps. In each of them, every vertex is processed and can be repositioned by the numerical solver. The vertices that lie on the mesh surface are treated as fixed and are not updated.

The most time-consuming operation called $VertexRepositioning$ moves free vertices (V) of an input mesh (M). It iterates an *inner loop* (lines 9-24) while an extreme of the objective function (K) is being reached by a numerical method (NM). K is constructed after determining the above mentioned VrPA parameters: A, f and q (lines 17-20). $LogicFunction$ uses a termination criterion (TC) to stop the algorithm (line 30). The *outer loop* (lines 30-37) is iterated in the $Main$ 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 worst mean-ratio quality metric of the mesh [6]. At the end of the algorithm, an optimized mesh is obtained.

4 Experimental setup

Algorithm 1 was used to compare the above mentioned VrPA configurations. The following paragraphs describe details of the implementation.

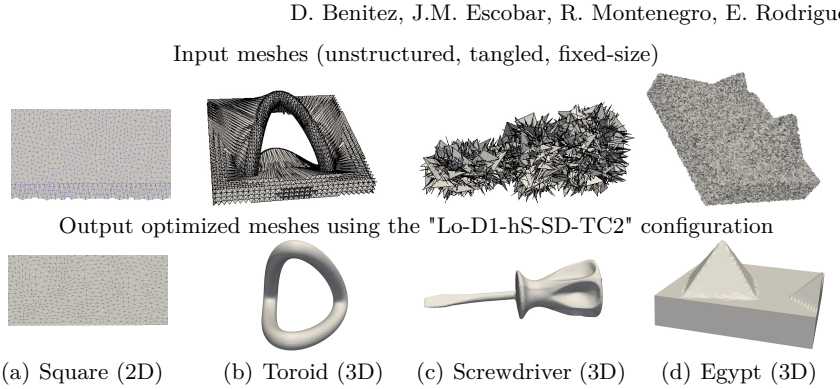
Software framework. We developed complete programs that include double-precision floating-point data structures and $PAPI$ functions for hard-

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 optimization method (NM)
3: #define termination criteria: TC = LogicFunction(Q_min, ΔQ̄, ΔQ_min)
4: #define constants: τ = 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)
5: N_e ← 0, N_f ← 0 ▷ Global variables: element evaluations (N_e), objective function evaluations (N_f)
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(minimizing) or ΔK ≥ τ(maximizing)) and m ≤ N_mII 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:        N_f += 1          ▷ Number of evaluations of the objective function and its derivative
24:        m += 1                                  ▷ Number of inner iterations
25:    return X̂                                    ▷ Output: updated coordinates of free vertices
26: procedure main( )
27:    ▷ Read the vertex and element information of M mesh
28:    Q_min ← GlobalMeasures(M)                   ▷ Minimum quality of input mesh
29:    ▷ Initiation : ΔQ̄ = 10^6, ΔQ_min = 10^6, k = 0 (loop index)
30:    while TC ≠ true and k ≤ N_mOI do            ▷ Mesh/Outer loop
31:        if A = Gl then                          ▷ Gl : all-vertex approach
32:            X_V ← VertexRepositioning(V, X_V, N_M)
33:        else                                    ▷ Lo : single-vertex approach
34:            for each free vertex v ∈ M do
35:                x_v ← VertexRepositioning(v, x_v, N_v)
36:        (Q_min, ΔQ̄, ΔQ_min)← GlobalMeasures(M)
37:        k += 1                                  ▷ Number of mesh/outer iterations
38:    ▷ Output: file with information of optimized M mesh
```

ware performance monitoring [4]. The source code includes some C++ classes and methods from the *Mesquite* framework [3]; all of them were modified to evaluate the computer performance and workload of vertex repositioning algorithms. The *Mesquite* method **TU** was also modified as previously explained in Section 2.2. We created new C++ classes and methods to support **hS** and **MQ** quality metrics, **log1**, **log2** and **inv** objective function formulations and **TC2** termination criteria (see Table 1). We used *gcc* 4.8.4 with `-O2` flag on *Linux* systems. For each VrPA configuration, we repeated the execution of the 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. All the mesh sizes were always fixed. The 2D mesh was obtained by using *Gmsh* tool [8], 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 [9]. All 3D meshes were obtained from a tool, called *The Meccano Method*, for adaptive tetrahedral mesh generation that tangles the mesh in one of its intermediate stages [12].



(a) Square (2D) (b) Toroid (3D) (c) Screwdriver (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, which 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 number of elements of local patch (N_v)	6.00	21.27	21.51	23.96
Average edge length	0.02	1.49	8.96	14.30
Standard deviation of the edge length	0.02	1.86	6.22	5.79
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

Figure 2 shows some convergence plots that represent the worst mean-ratio quality metric and the number of inverted elements versus the number of mesh iterations. They were obtained when the Lo-D1-hS-SD-TC2 configuration was used to optimize two of the meshes. The rest of "...-TC2" configurations and meshes reported in this paper exhibit similar convergence behaviors. Two consecutive stages can be identified in each optimization problem. Firstly, an untangling stage in which the inverted elements decrease over time to zero. Secondly, a smoothing stage where the worst mean-ratio quality metric increases from zero to a stable value, when TC2 is met. The convergence behaviors of "...-TC1" configurations exhibit only the untangling stage.

Processor Cores. Numerical experiments were conducted on three computers with different processor models. One of them is Intel Xeon E5645 that integrates 6 Westmere-EP cores whose clock speed is 2.4 GHz and are connected to 48 GB of DDR3/1600 MHz. Another processor is Intel Xeon E5-2670 that integrates 8 Sandy Bridge-EP cores whose clock speed is 2.6 GHz and are connected to 32 GB of DDR3/1333 MHz. The third processor is Intel Xeon E5-2690V4 that integrates 14 Broadwell cores whose clock speed is 2.6 GHz and are connected to 256 GB of DDR4/2400 MHz. During the experiments, the compute nodes were not shared among other user-level workloads. Additionally, multithreading and Turbo Boost were disabled.

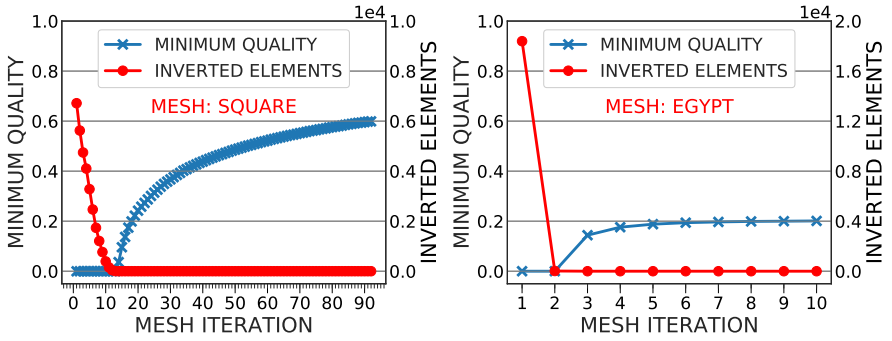


Fig. 2 Convergence plots obtained when the Lo-D1-hS-SD-TC2 configuration was employed to optimize the Square (left) and Egypt (right) meshes.

5 Performance comparison of VrPA algorithms

The execution time of Algorithm 1 varies widely, usually with a trade-off between different VrPA parameters. This can be seen in Figures 3 and 4 that show the results of a performance evaluation using 136 configurations and two processors (E5645, E5-2670). Those configurations that not appear in these figures produce tangled meshes.

Results are grouped by benchmark mesh. Each graph distinguishes the convergence criterion established and the processor used. The goal of the TC1 criterion is to produce a mesh with no inverted elements. The other criterion, TC2, is met when the mesh is optimum, i.e., the increase in both the worst and average mean-ratio quality metrics after two successive mesh iterations are below a certain threshold. These graphs include the average and minimum mean-ratio quality metrics of output meshes. Note that there is more variability in the execution time than in the quality metrics.

Table 3 shows the fastest VrPA configuration for each mesh when the TC1 convergence criterion was established. The main results of this partial analysis are as follows: (1) the minimum quality metrics (Q_{min}) of output meshes vary significantly; (2) global approaches (Gl−...) frequently achieve larger Q_{min} than local approaches (Lo−...); (3) the average quality metrics ($\overline{Q}$) of output meshes exhibit much less variability than the minimum values; (4) the fastest algorithm always uses Gl approach, hS metric and SD solver; (5) the fastest configurations on all processors are the same; (6) the solver with superior performance is not always the same; however, the performance behavior of the SD solver is frequently superior to the CG solver.

Table 4 shows the fastest configuration for each mesh when the TC2 criterion was established. The main conclusions of this another analysis are: (1)Q_{min} is much larger and exhibits less variability than for mesh untangling (TC1); (2) the algorithms that achieve the largest Q_{min} always use local approach (Lo−...) and barrier formulation (log1, log2, inv); (3) $\overline{Q}$ exhibits slightly larger values and less variability than for mesh untangling; (4) there are no significant differences in Q_{min} and $\overline{Q}$ between SD and CG solvers,

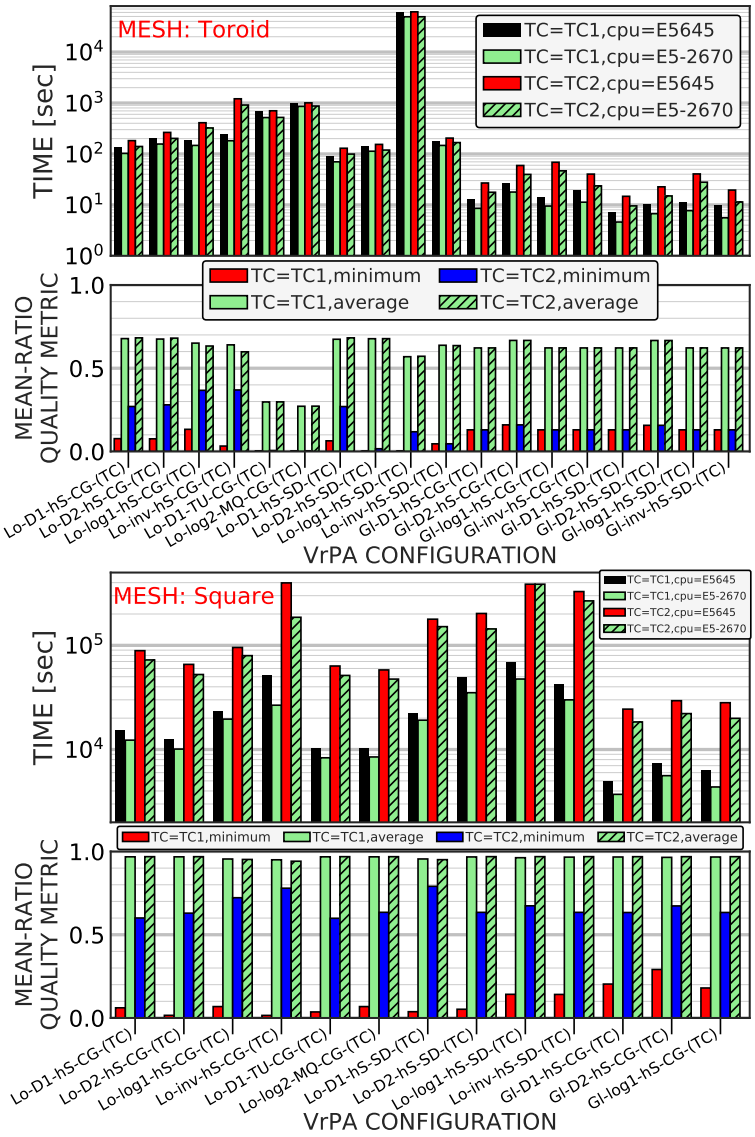


Fig. 3 Execution times and quality metrics for the Square and Toroid meshes.

except for the Toroid mesh whose results obtained with CG solver are superior; (5) the fastest algorithm always uses the SD solver and D1 formulation; additionally, the hS quality metric takes more times the first place in the rank ordering of performance than the other formulations and quality metrics; (6) the fastest configurations on both processors are again coincident; (7) the ratio of the time required by the configuration with the largest Q_{min} to the lowest time is larger than the ratio of respective Q_{min} except for the Screwdriver mesh; this means that the highest performance configurations

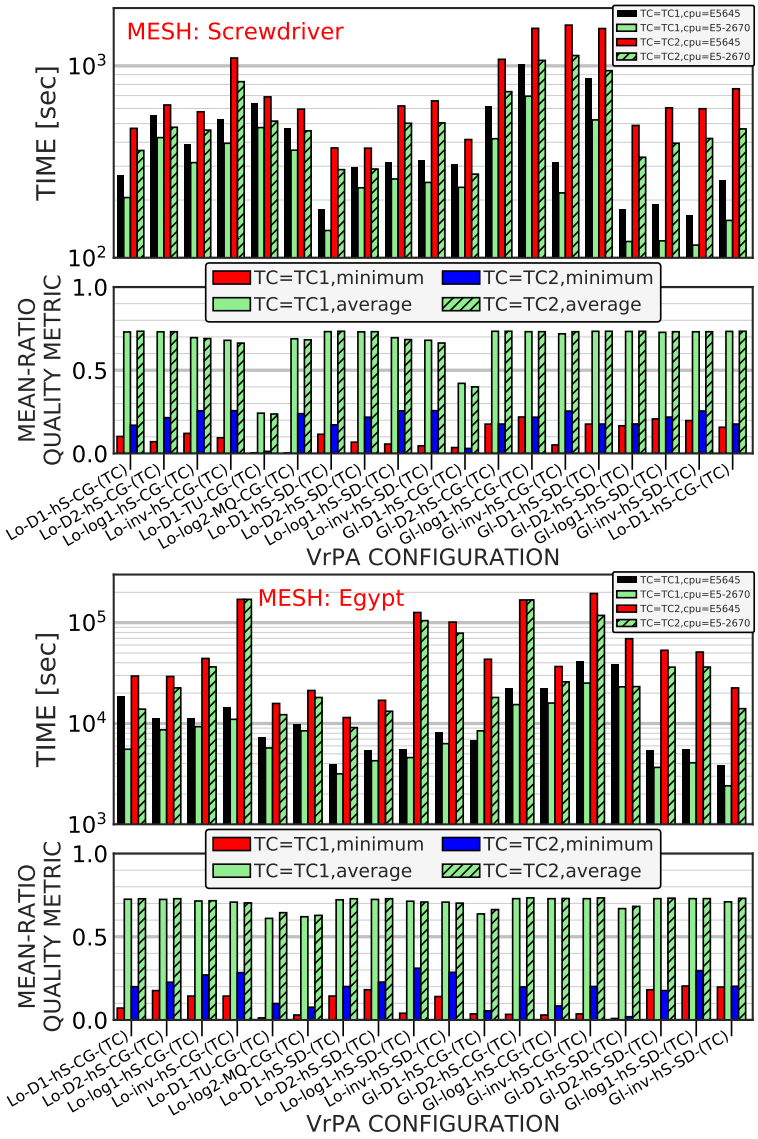


Fig. 4 Execution times and quality metrics for the Screwdriver and Egypt meshes.

may be the best choices for obtaining approximate solutions in the smallest amount of time.

6 Global workload analysis of VrPA algorithms

Some authors use the number of elements (mesh size) as a workload measure to estimate the execution time of VrPA algorithms [15, 17]. However, this can be accurately done only when the configuration is fixed and the total

Table 3 VrPA configurations with lowest runtimes for the TC1 convergence criterion. N_e is the number of mesh element evaluations that were required by each configuration.

Benchmark mesh	**Fastest VrPA configuration**					
	Configuration	**CPU time [s]**		N_e	$\frac{largest\ CPU\ time}{CPU\ time}$	
		E5645	**E5-2670**		**E5645**	**E5-2670**
Square	Gl-D1-hS-SD-TC1	4.96 10^3	3.87 10^3	4.42 10^9	13.8	12.8
Toroid	Gl-D1-hS-SD-TC1	7.4	4.61	8.73 10^6	8320.9	10691.2
Screwdriver	Gl-log1-hS-SD-TC1	1.66 10^2	1.17 10^2	1.84 10^8	6.1	6.0
Egypt	Gl-inv-hS-SD-TC1	3.86 10^3	2.41 10^3	3.38 10^9	10.8	10.4

Table 4 VrPA configurations that meet the TC2 convergence criterion and achieve the lowest CPU time or the largest Q_{min} (worst mean-ratio quality metric).

Mesh	**Comparison criterion**	**Best configuration**	**CPU time (sec)**		Q_{min}	$\frac{time\ largest\ Q_{min}}{lowest\ time}$	
			E5645	**E5-2670**		**E5645**	**E5-2670**
Square	lowest t_{CPU}	Gl-D1-hS-SD-TC2	2.44 10^4	1.84 10^4	0.633	7.3	8.2
	largest Q_{min}	Lo-log1-hS-SD-TC2	1.78 10^5	1.51 10^5	0.791		
Toroid	lowest t_{CPU}	Gl-D1-hS-SD-TC2	1.47 10^1	9.56	0.13	82	95
	largest Q_{min}	Lo-inv-hS-CG-TC2	1.21 10^3	9.12 10^2	0.369		
Screwdriver	lowest t_{CPU}	Lo-log2-MQ-SD-TC2	3.04 10^2	2.73 10^2	0.03	3.6	3.0
	largest Q_{min}	Lo-inv-hS-CG-TC2	1.1 10^3	8.27 10^2	0.257		
Egypt	lowest t_{CPU}	Lo-D1-hS-SD-TC2	1.15 10^4	9.11 10^3	0.201	11	12
	largest Q_{min}	Lo-log1-hS-SD-TC2	1.26 10^5	1.05 10^5	0.311		

numbers of inner and outer iterations of the algorithm are both fixed. In this case, the execution time is directly proportional to the size of the mesh.

For the algorithms where the VrPA parameters are free and the number of iterations is variable and based on convergence criteria, this proportionality is not evident as can be seen in Figure 5(a). This figure shows a graph of mesh size versus execution time that was derived from the results of the above-described experiments for mesh untangling. Each point represents one of 68 VrPA configurations that successfully untangle a fixed-size test mesh (see the "...-TC1" configurations in Figures 3 & 4). The correlation coefficient between time and mesh size taken in the linear scale is r=0.44.

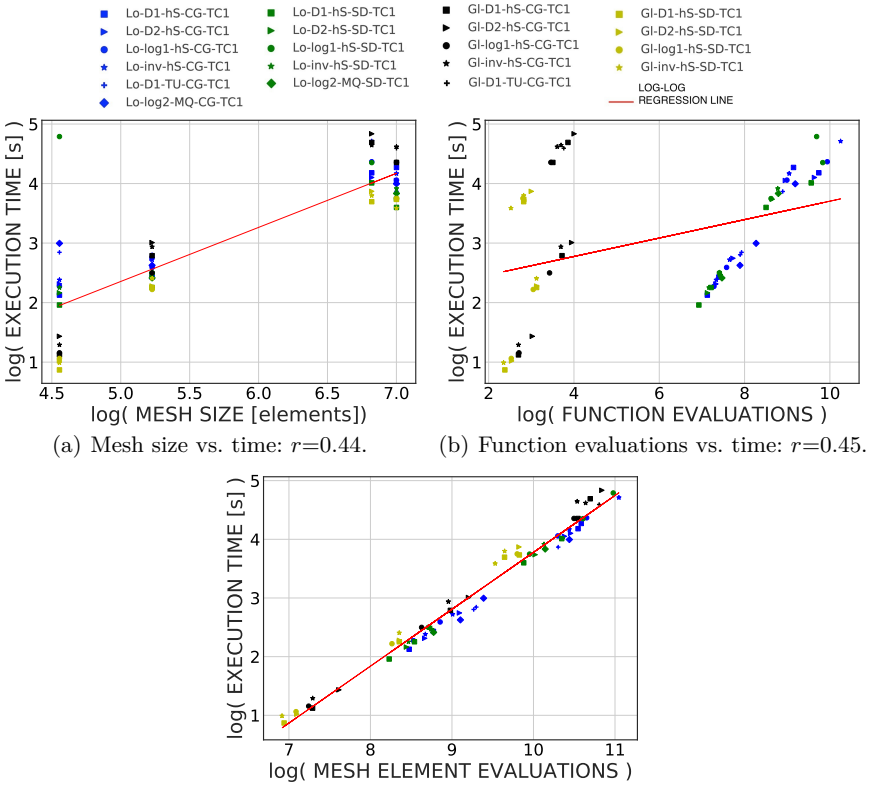
The number of evaluations of the objective function and its derivative (N_f) has also been used as a workload measure in numerical algorithms [18]. Line 23 in Algorithm 1 was employed to count the number of evaluations of the objective function and its derivative. Figure 5(b) shows a graph of N_f versus time for the same "...-TC1" configurations. In this case, the correlation coefficient between time and N_f taken in the linear scale is r=0.45.

N_e in Algorithm 1 is called *number of mesh element evaluations* and measures the number of evaluations of the element quality metric and its deriva-

tive. This measure involves computing the separable but not independent parts of objective function evaluations. Although not exactly the same definition, the mesh element evaluation is slightly similar to the *concurrent function evaluation step* defined in [18] for identifying parallelism opportunities in finite difference gradients.

Figure 5(c) shows a graph of N_e versus execution time for the "...-TC1" configurations. In this case, the correlation coefficient between time and N_e taken in the linear scale is r=0.93. Therefore, execution time is more directly proportional to the number of mesh element evaluations than the mesh size or the number of objective function evaluations. It is important to note that N_e is not very intrusive and depends not only on the problem size but also on the numbers of inner and outer iterations required to meet the convergence criteria. Thus, we will use N_e as a workload measure for VrPA algorithms in a new performance model that is proposed below.

Some of the fastest configurations that were evaluated in Section 5 achieve the highest performance because they need fewer element evaluations than



(a) Mesh size vs. time: r=0.44.

(b) Function evaluations vs. time: r=0.45.

(c) Element evaluations vs. time: r=0.93.

Fig. 5 Scalability of the algorithms for mesh untangling (convergence criterion: TC1) using different workload measures, the E5645 processor and the four benchmark meshes. r is the correlation coefficient taken in the linear scale.

the others (see N_e in Table 3). This was the case for all of our test meshes except Toroid when the TC1 criterion was established. To untangle the Toroid mesh, Gl-inv-hS-SD-TC1 configuration needs less N_e ($8.23\ 10^6$) than the fastest configuration ($8.73\ 10^6$). However, its time per element evaluation is sufficiently larger than the fastest configuration such that the time to convergence is not the lowest.

In summary, the performance of VrPA algorithms depends on the balance between two factors: the global workload measured in number of element evaluations and the time per element evaluation. The first factor 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. Moreover, the number of evaluations also depends on the characteristics of the input mesh such as the quality of elements or the number both of free vertices and of elements of each patch. The second factor is affected also by all these algorithmic, software and mesh aspects in addition to the computer hardware.

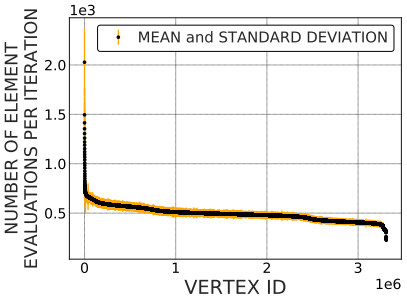
7 Analysis of the workload required by a free vertex

For a deeper understanding of VrPA algorithms, this section analyzes the workload required by vertices. Figure 6 shows the average and standard deviation of the number of element evaluations (N_e) that were needed by each free vertex in every outer iteration for a selection of configurations. Note that vertices are sorted by number of element evaluations from largest to smallest.

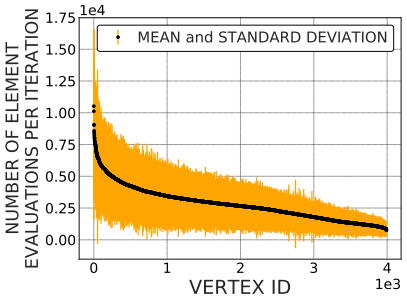
As it can be seen, the average workload per vertex of algorithms that use local approach (Lo$-\dots$) is unbalanced, i.e., there is a large range of workloads (see the black lines in Figures 6(a), 6(b), 6(d)). The maximum of the range can be one order of magnitude larger than the minimum. The variance of the number of element evaluations per vertex over different mesh iterations is also variable. It can be as large as the mean value (see Figure 6(b)), or close to zero (see Figures 6(a), 6(d)).

The algorithms that use a global approach (Gl$-\dots$) exhibit very different workload behaviors. The average and variance of the number of element evaluations per free vertex and mesh iteration are both constants (see Figure 6(c)). This is due to that each free vertex is repositioned after the displacement directions of all vertices have been obtained. A common way to do this is to evaluate all elements in each evaluation of the single objective function and its derivative (see lines 12-15,18-20 in Algorithm 1).

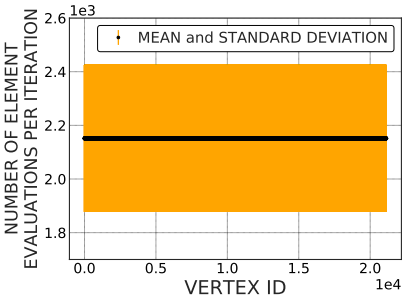
In contrast, a local algorithm uses many independent objective functions, each of them determines the displacement direction of a single vertex. The number of element evaluations required for every objective function is variable. Thus, each free vertex causes a different computer workload. As the vertices are repositioned in series and the execution time was demonstrated in Section 6 that is highly correlated with the number of element evaluations, each vertex contributes to the total execution time differently.



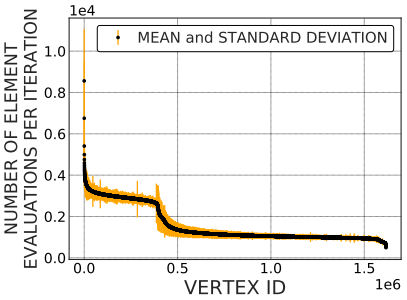
(a) Mesh: Square, VrPA: Lo-D2-hS-SD-TC1, 14 mesh iterations, 5.1 10^{-7}sec/evaluation.



(b) Mesh: Toroid, VrPA: Lo-D1-hS-CG-TC1, 14 mesh iterations, 4.2 10^{-7}sec/evaluation.



(c) Mesh: Screwdriver, Gl-log1-hS-SD-TC1, 4 mesh iterations, 9.5 10^{-7}sec/evaluation.



(d) Mesh: Egypt, VrPA: Lo-D1-hS-SD-TC1, 3 mesh iterations, 5.2 10^{-7}sec/evaluation.

Fig. 6 Mean and standard deviation of the number of element evaluations per free vertex and mesh iteration. These results were obtained using the TC1 termination criterion for mesh untangling, the E5645 processor, a selected group of VrPA configurations and all the benchmark meshes.

8 Sequential performance model

Performance models combine methods that provide expectations on performance and instrumentation tools that find parameters with empirical measurements [1]. Taking the findings of Section 6, we use a simple one-parameter model to understand the performance of sequential VrPA algorithms,

$$t_{CPU} = \alpha \; N_e \tag{4}$$

where t_{CPU} denotes the execution time, N_e denotes the number of mesh element evaluations and α denotes the model parameter that represents the time per element evaluation. Equation 4 assumes that the computation time is much larger than the 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 5(c)). This effect can be justified by our model.

Equation 4 calculates the time by multiplying two factors. A VrPA configuration can need larger number of element evaluations than other: $N_{e,1} > N_{e,2}$. However, the second algorithm can evaluate each element in more time than the first algorithm: $\alpha_1 < \alpha_2$. This is what happens to some configurations.

Taking the execution times and the respective numbers of element evaluations for the TC2 convergence criterion, we calculated the time per element evaluation for each configuration using Equation 4. These results for the Screwdriver mesh are shown in Figure 7(a). We found similar results when the other benchmark meshes were optimized. As it can be seen, each configuration causes a different time per element evaluation. Combining the results shown in Figures 5(c) and 7(a), it can be demonstrated that the time per element evaluation is not correlated with the number of element evaluations (E5645 processor: r=-0.04). Therefore, the smallest product of the two factors of Equation 4 determines what algorithm achieves the best performance.

8.1 Model application and accuracy

To check the accuracy of our one-parameter model, we used the 68 configurations of Section 5 that meet the TC1 convergence criterion. Note that we let all the VrPA parameters and input meshes vary freely except the termination criterion. Randomly chosen, half of the VrPA configurations were used to derive α. We calculated for each configuration the ratio of execution time to the number of element evaluations: $\alpha = t_{CPU}/N_e$. After averaging the results, $\overline{\alpha}$ was 6.7 10^{-7} and 4.9 10^{-7} [sec/element] when E5645 and E5-2670 processor cores were used, respectively. The relative errors in estimating the execution times were obtained with the other half of configurations, $\overline{\alpha}$ and Equation 4. The average relative errors were 0.27 and 0.34 for the E5645 and E5-2670 processor cores, respectively.

We extended this accuracy analysis by setting free only one of the five VrPA parameters (see Table 1). The input mesh was considered as a sixth parameter. For this study, we analyzed a total of 136 configurations. From these configurations, we took groups that have one free and five fixed parameters. Due to the possible choices of each parameter, the number of configurations included in every group was between two and five. For each group, we obtained the mean time per element evaluation ($\overline{\alpha}$). Then, we compared the time provided by our model (Equation 4) with the real execution time for each configuration in every group. The mean and maximum relative errors of our model using the above-mentioned processors are shown in Table 5.

The errors are caused by the variability in the values of α that are obtained for the configurations that constitute each group. The largest variability about the mean occurs in the groups that combine all-vertex and single-vertex algorithms where the rest of VrPA parameters are the same. Our model fits best when it is applied to a single VrPA algorithm and mesh. In this case, the same α can be used to justify accurately the execution time for different convergence criteria. Moreover, our results indicate that the model parameter

(α) depends on the processor architecture. Thus, α must be recalculated if the processor changes.

Table 5 Mean and maximum relative errors in estimating the execution times of VrPA algorithms on E5645 and E5-2670 processors.

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	0.04
Mesh	0.07	0.20	0.09	0.41

9 Architectural analysis

As it can be seen in Figures 3 and 4, the E5-2670 processor core outperforms the E5645 core. Our performance model (Equation 4) indicates that it is due to smaller time per element evaluation (α) as the number of element evaluations (N_e) of a VrPA algorithm is the same on both processor cores.

Figure 7(a) shows the effects of three processor microarchitectures on the time per element evaluation. The main conclusions drawn from the results presented in this figure are the following: (1) the E5645 core spends on average 1.4 and 2.1 times more execution time per element evaluation than the E5-2670 and E5-2690V4 cores, respectively; (2) global methods (Gl−...) always require more execution time per element evaluation than local methods (Lo−...); (3) the configurations that employ the SD solver also spend more time in each element evaluation than the configurations that use CG solver if the rest of VrPA parameters are the same.

We have also investigated the causes of this variability in execution time per element evaluation at the hardware level. Our programs were instrumented using the PAPI programming interface [19] and informative performance counters of Intel cores [10]. In this investigation, we also used the classic CPU performance equation that relates the execution time to instruction count, clock cycles-per-instruction (CPI) and clock rate [13],

$$t_{CPU} = \frac{InstructionCount \;\; CPI}{ClockRate} \tag{5}$$

CPI is a performance metric that mainly depends on the hardware microarchitecture, instruction-set architecture, compiler and programming language [13]. In our experiments, instruction-set architecture, compiler and programming language remained the same.

Figures 7(b,c,d) show measurements of three performance metrics per element evaluation for the Screwdriver mesh and VrPA configurations that meet the TC2 convergence criterion. These metrics are: instruction count (retired instructions), CPI and miss rate of the L1 data cache. The configurations have

been sorted by time per element evaluation from the smallest to the largest numerical value. So, the correlation between time per element evaluation and the measurements provided by the hardware counters can be perceived.

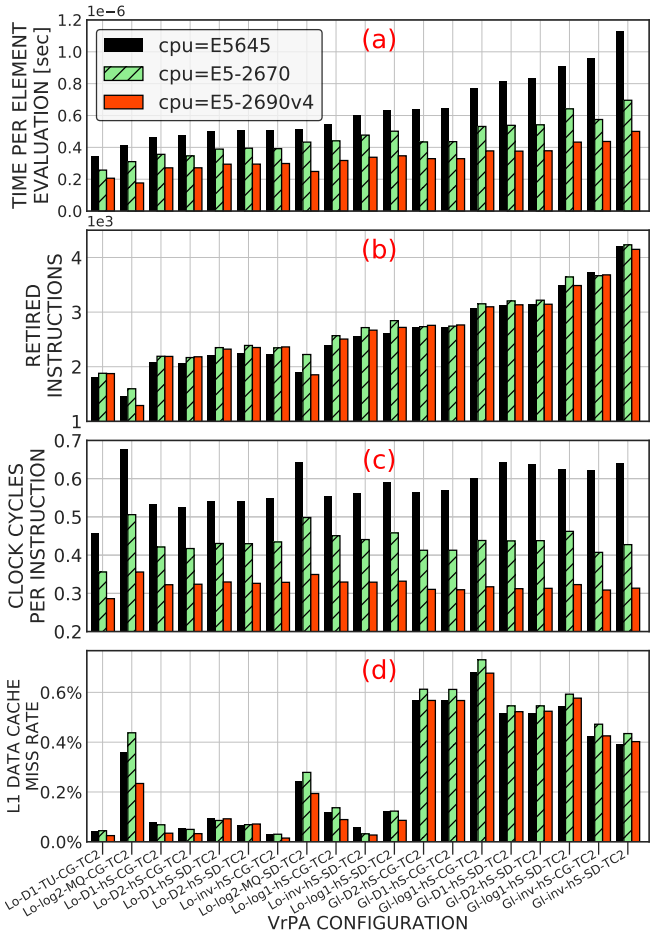


Fig. 7 Performance metrics per mesh element evaluation:

(a) execution time

(b) total retired instructions

(c) clock cycles per instruction (CPI)

(d) miss rate of the Level-1 data cache

Mesh: Screwdriver

The main conclusions of the analysis of these performance metrics per element evaluation are: (1) longer time per element evaluation is frequently caused by a larger number of retired instructions per evaluation when the same microarchitecture is analyzed (see Figure 7(b)); note that global methods (Gl−...) execute more instructions than local methods (Lo−...); (2) when the same VrPA configuration is evaluated on the three processors, the number of retired instructions per evaluation is approximately the same, which is to be expected because we are using the same program, compiler and instruction-set; (3) for each configuration, the CPI of the E5-2690V4 core is always lower than the CPI of the E5-2670 core whose CPI is lower than the CPI of the E5645 core (see Figure 7(c)); this is to be expected from newer generations of microarchitectures given that fewer number of stall cycles per

instruction is frequently a hardware design goal for commodity processors; (4) because of the ratio of clock rates of these three processors is lower than 1.08 and the instruction counts are similar, most of the performance advantage comes from a much lower CPI for the newer processors; on average, the ratio between the CPIs of E5-2670 and E5645 is 1.22, and the ratio between the CPIs of E5-2690V4 and E5645 is 2.3; (5) global configurations always cause larger miss rates of the L1 data cache than local configurations (see Figure 7(d)), although the impact on the CPI is not very significant.

10 Conclusions

We have studied the performance and workload of vertex repositioning algorithms (VrPA) for simultaneous mesh untangling and smoothing. The influence on performance and workload of a large number of VrPA based on five parameters using triangle and tetrahedral meshes has been investigated. Among the choices for possible VrPA parameters, two new objective function formulations have been proposed. The largest worst mean-ratio quality metric for each benchmark mesh was obtained using one of these formulations. Additionally, a performance model for sequential VrPA algorithms has been proposed. This model involves a new workload measure called *number of mesh element evaluations* that is independent of the processor if numerical precision, compiler, and program are the same. We have shown that the execution times of VrPA methods are more proportional to the number of element evaluations than mesh size or the number of objective function evaluations. The other factor of the model, the time per element evaluation, is more affected by the processor and objective function approach than the objective function formulation, the quality metric, numerical solver, convergence criteria or mesh. The performance of VrPA methods has been compared using three processor cores with different microarchitectures. Most of the advantage of newer processors come from smaller values of the CPI performance metric.

This paper has also shown that the workload per vertex of a local optimization algorithm is very unbalanced. Using this finding, we have devised a new mesh partitioning approach [2]. Our study methodology may be applied to meshes with elements that are different from triangles and tetrahedra. For instance, some vertex repositioning methods improve the quality of hex-meshes after solving an optimization problem [11, 14]. They employ iterative solvers to minimize different objective functions composed of terms that measure the element qualities. However, the validity of our findings across these other methods has to be examined.

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. One of the

computers used in this work 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. Benítez, J.M. Escobar, R. Montenegro, E. Rodríguez: Parallel Performance Model for Vertex Repositioning Algorithms and Application to Mesh Partitioning. In: Proc. 27^{th} Int. Meshing Roundtable, 2018.
3. M. Brewer, L. Diachin, P. Knupp, T. Leurent, D. Melander: The Mesquite Mesh Quality Improvement Toolkit. In: Proc. 12^{th} Int. Meshing Roundtab., 239-250, 2003.
4. S. Browne, J. Dongarra, N. Garner, K. London, P. Mucci: A Scalable Cross-Platform Infrastructure for Application Performance Tuning Using Hardware Counters. In: Proc. Supercomputing, Article 42, IEEE Comp. Soc., 2000.
5. Y. Che, L. Zhang, C. Xu, Y. Wang, W. Liu, Z. Wang: Optimization of a parallel CFD code and its performance evaluation on Tianhe1A. Computing and Informatics, 33(6):1377-1399, 2015.
6. L. Diachin, P. Knupp, T. Munson, S. Shontz: A Comparison of Inexact Newton and Coordinate Descent Mesh Optimization Techniques. In: Proc. 13^{th} Int. 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. C. Geuzaine, 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.
9. P. Knupp: Updating meshes on deforming domains: An application of the target-matrix paradigm. Commun. Num. Method Eng., 24:467-476, 2007.
10. D. Levinthal: Performance Analysis Guide for Intel Core i7 Processor and Intel Xeon 5500 Processors. Intel, 2014.
11. M. Livesu, A. Sheffer, N. Vining, M. Tarini: Practical hex-mesh optimization via edge-cone rectification. ACM Trans. Graph., 34, 4, Article 141, 2015.
12. 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.
13. D.A. Patterson, J.L. Hennessy: Computer Organization and Design. The Hardware/Software Interface. ARM edition. Morgan Kaufmann Publishers Inc., 2017.
14. E. Ruiz-Girones, X. Roca, J. Sarrate, R. Montenegro, J.M. Escobar: Simultaneous untangling and smoothing of quadrilateral and hexahedral meshes using an object-oriented framework. Advances in Engineering Software, 80, 12-24, 2015.
15. S.P. Sastry, S.M. Shontz: Performance characterization of nonlinear optimization methods for mesh quality improvement. Eng. with Computers, 28:269-286, 2012.
16. S.P. Sastry, S.M. Shontz: A parallel log-barrier method for mesh quality improvement and untangling. Eng. with Computers, 30(4):503-515, 2014.
17. S.P. Sastry, S.M. Shontz, S.A. Vavasis: A log-barrier method for mesh quality improvement and untangling. Eng. with Computers, 30(3):315-329, 2014.
18. R.B. Schnabel: Concurrent Function Evaluations in Local and Global Optimization. CU-CS-345-86. Comp. Science Tech. Rep. 332. Univ. Colorado, Boulder, 1986.
19. D. Terpstra, H. Jagode, H. You, J. Dongarra: Collecting Performance Data with PAPI-C. In: Tools for High Performance Computing 2009, Springer, 157-173, 2010.
20. C.S. Verman, K. Suresh: Towards FEA over tangled quads. Procedia Engineering, 82:187-199, 2014.