Advances in Engineering Software 157–158 (2021) 103006

Contents lists available at ScienceDirect

Advances in Engineering Software

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

Research paper

Multi-GPU acceleration of large-scale density-based topology optimization

David Herrero-Pérez [*,a,b], Pedro J. Martínez Castejón [b]

[a] Computational Mechanics and Scientific Computing Group, Technical University of Cartagena, Cartagena, Murcia 30202, Spain
[b] Technical University of Cartagena, Campus Muralla del Mar, Cartagena, Murcia 30202, Spain

ARTICLE INFO

Keywords:
Topology optimization
GPU computing
Multi-GPU systems
Finite element analysis
Aggregation AMG

ABSTRACT

This work presents a parallel implementation of density-based topology optimization using distributed GPU computing systems. The use of multiple GPU devices allows us accelerating the computing process and increasing the device memory available for GPU computing. This increment of device memory enables us to address large models that commonly do not fit into one GPU device. The most modern scientific computers incorporate these devices to design energy-efficient, low-cost, and high-computing power systems. However, we should adopt the proper techniques to take advantage of the computational resources of such high-performance many-core computing systems. It is well-known that the bottleneck of density-based topology optimization is the solving of the linear elasticity problem using Finite Element Analysis (FEA) during the topology optimization iterations. We solve the linear system of equations obtained from FEA using a distributed conjugate gradient solver pre-conditioned by a smooth aggregation-based algebraic multigrid (AMG) using GPU computing with multiple devices. The use of aggregation-based AMG reduces memory requirements and improves the efficiency of the interpolation operation. This fact is rewarding for GPU computing. We evaluate the performance and scalability of the distributed GPU system using structured and unstructured meshes. We also test the performance using different 3D finite elements and relaxing operators. Besides, we evaluate the use of numerical approaches to increase the topology optimization performance. Finally, we present a comparison between the many-core computing instance and one efficient multi-core implementation to highlight the advantages of using GPU computing in large-scale density-based topology optimization problems.

1. Introduction

Topology optimization techniques allow us to find the optimal distribution of material within a design domain such that a cost function is minimized subject to a set of constraints. Such methods provide us with innovative and high-performance conceptual designs at the early stages of the design process without assuming any prior structural configuration. For these reasons, engineering designers adopt topology optimization techniques as a powerful tool that allows them to design lightweight and optimized structures in a broad spectrum of industries [1]. We can broadly classify the topology optimization methods into three main categories depending on the representation used to describe the shapes they involve: *Eulerian, Lagrangian*, and *density-based* methods. Eulerian methods use an implicit representation of the structural boundary that is modified to optimize the design. Some examples of these implicit representations are the level-set function tracked in the Level-Set Method (LSM) [2] and the interfacial dynamics evolved in the phase field methods [3]. Lagrangian methods use an explicit representation of the structural shape employing a computational mesh or CAD model [4]. Density-based methods operate on a fixed mesh of finite elements to find the optimal void/solid material distribution that minimizes an objective function. The homogenization method [5] and the Solid Isotropic Material Penalization (SIMP) method [6,7] are some examples of the most popular density-based topology optimization approaches. The latter has become the most popular and implemented method in commercial software of topology optimization, probably due to its simplicity and feasibility. The density-based topology optimization method SIMP is the approach that we adopt in this work.

The mesh resolution is of paramount importance in the topology optimization process. We need fine tessellation for capturing the geometric details of the design, obtaining more optimized designs by increasing the number of design variables. The use of high-resolution finite element models gives rise to a large system of equations, which should be solved efficiently to make the process feasible, both in computing time and computational resource requirements terms. This problem is a well-known computational challenge due to the constant-

* Corresponding author.
E-mail addresses: david.herrero@upct.es (D. Herrero-Pérez), pedro.castejon@upct.es (P.J. Martínez Castejón).

https://doi.org/10.1016/j.advengsoft.2021.103006
Received 28 December 2020; Received in revised form 27 March 2021; Accepted 7 April 2021
Available online 29 April 2021

increasing in the required fidelity and complexity of finite element models [8].

Multiresolution topology optimization techniques decouple analysis and design discretizations. They improve the computational performance by using a coarse discretization for the analysis and a fine discretization for the design variables [9,10]. However, the iterative updates of the topology optimization design variables depend on the analysis results, and thus some Adaptive Mesh Refinement (AMR) technique is needed to obtain meaningful benefits, which depend on the complexity of the design [11]. The use of approaches for increasing the efficiency of the topology optimization process is also rewarding. Some examples are the rescaling of large systems of equations to reduce the ill-conditioning [12], the use of low accurate approximations [13] of the analysis solution, and the use of efficient preconditioners [14]. We also have to mention using reanalysis techniques that avoid the full analysis of the modified design in the optimization procedure. Some examples are the use of approximate reanalysis [15] by only solving the system of equations at an interval of iterations and approximating the solution at other iterations of the nested topology optimization process, the use of iterative reanalysis in moving morphable component-based topology optimization [16] for improving the computational performance of the static analysis, and the use of isogeometric reanalysis in structural optimization modifying the control points of the design [17]. Finally, High-Performance Computing (HPC) has also been used to address the computationally intensive tasks of the topology optimization process using multi-core computing efficiently [18–22].

The use of graphics cards for scientific computing is becoming an integral part of modern high-performance systems. We can find these massively parallel many-core architectures in most modern supercomputers [23] and high-performance desktop workstations. However, the proper exploitation of General-Purpose computing on Graphics Processing Units (GPGPUs) [24] requires the adoption of algorithms with data parallelism. Another fundamental point is the limited device memory of Graphics Processing Units (GPUs), and thus the combination of GPU computing with function parallelism in distributed GPU systems will allow us to alleviate such a constraint [25].

The use of massively parallel many-core architectures to address topology optimization problems is growing in popularity last decade. The motivation is to reduce the computational burden of the topology optimization process by commonly exploiting data locality. The early work of Wadbro and Berggren [26] proposed the use of GPU computing for the evaluation of high-resolution finite element models in heat conduction topology optimization. They adopted a Preconditioned Conjugate Gradient (PCG) method with an assembly-free element-wise implementation to reduce the device memory requirements. Schmidt and Schulz [27] proposed a nodal-wise assembly-free PCG implementation using GPU computing for solving the elasticity problems at the iterations of the minimization of the structural compliance problem using the SIMP method on a regular grid of hexahedral finite elements. The proposal achieved significant speedups in the matrix-vector operations of the iterative solver using shared memory. They grouped the shared memory operations in successive 2D slices of the stencil-based computation, which increases the GPU performance performing the matrix-vector operations [28]. Another strategy for improving the GPU performance is the reduction of the grain-size in the assembly-free GPU implementation [29].

The use of GPU computing in geometric multigrid methods has also shown to be rewarding. These approaches are with the most efficient and popular techniques to solve large linear systems of equations. Krylov subspace methods commonly use these methods as a preconditioner in the context of structural mechanics. The main handicap of implementing geometric multigrid methods using GPU computing is the amount of memory for storing the coefficient matrix and interpolation operators at the different levels. For this reason, Dick et al. [30] proposed an efficient nodal-wise matrix-free GPU implementation of the geometric multigrid method with stencil computation for solving elasticity problems with the finite element method. They exploit the regularity of a Cartesian grid to obviate the storage of the coefficient matrix, replacing the use of memory by "on-the-fly" calculations performed in parallel using GPU computing. The stencil computation also allows exploiting data locality. In particular, enabling coalescing memory access into single memory transactions and implementing efficient "on-the-fly" stencil-based grid transfer operators.

Several works have followed this strategy to achieve high-performance GPU computing in the topology optimization of structural mechanics problems [31–34]. However, the adoption of ersatz material approximation with a regular Cartesian grid induces errors in the FEA analysis [2]. We use such analysis to evaluate the objective function of the optimization algorithm. This approximation consists of the use of a bi-material equation interpolating in the boundary of finite elements. This strategy usually requires the refinement of the regular Cartesian grid to achieve a reasonable error estimation. However, the tessellation cannot be too small since the problem becomes more and more badly conditioned as the discretization size tends to zero [35]. This lack of precision can be an obstacle to the optimization problem.

The number of finite elements using a regular Cartesian grid to represent the domain of the design space is higher than tessellating the CAD model using conventional methods. Conventional meshing techniques also provide suitable finite elements for the analysis and a higher degree of flexibility for implementing multiphysics problems. For these reasons, recent works make use of GPU computing to efficiently assemble and solve the system of equations of multiphysics using sparse-matrix representation in topology optimization [36]. Many investigations also evaluate the GPU acceleration of topology optimization using unstructured meshes, usually achieving low speedup in the solving stage using iterative methods [37]. The use of multiple GPUs to accelerate the solving and increase the device memory size using unstructured meshes and sparse-matrix representation is also studied by [38], focusing on the optimization of the communication strategy to achieve performance increases. We discard the use of direct solvers using GPU computing due to the memory constraints of such devices [39]. The Krylov subspace iteration method has a lower memory requirement, but the assembly process and storage of the coefficient matrix can exceed the device memory of one single GPU. The use of multiple graphics cards allows us to increase the amount of device memory available to address the problem and to achieve higher acceleration in demanding tasks. On the other hand, the adoption of techniques saving memory can increase the performance of GPU computing meaningfully.

Despite the promising potential of using multiple GPUs for addressing topology optimization problems, the use of such high-performance systems remains small. One example is accelerating the parallel evaluation of stochastic collation points in robust topology optimization problems using multiple GPUs in the same host [40]. The stochastic collocation approach is embarrassingly parallel, allowing evaluating each stochastic model by only one GPU device. Such an approach minimizes the communications to the calculation of stochastic moments. Nevertheless, the use of domain-decomposition methods [41] provides a higher degree of flexibility. Besides, it facilitates the exploitation of the computational capabilities of modern computing architectures.

This work aims to evaluate the use of multiple GPU devices in density-based topology optimation. The objective is two-fold: it accelerates the computing process and increases the available device memory. The latter allows us to solve large models that commonly do not fit on one GPU. It is well-known that the solving stage of the linear system of equations that evaluates the objective function dominates the overall speedup. Thus, the work focuses on testing suitable techniques to solve large systems of equations exploiting these massively parallel architectures. We also take into account the flexibility and easy integration of the methods adopted. In this sense, Algebraic Multigrid Methods (AMG) only use the information provided by the coefficient matrix, allowing their use as a "black-box" function in finite element codes. We adopt a distributed conjugate gradient solver using the memory available by

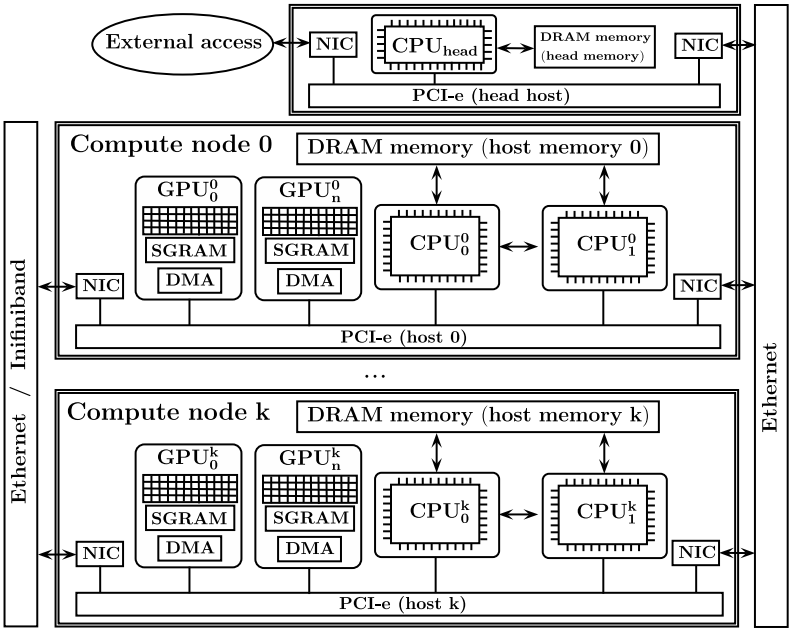


Fig. 1. Hardware architecture of GPU cluster and deployment connections.

each processor, including host and device memory of each graphics card. We also use a smooth aggregation-based AMG method for preconditioning the Krylov subspace iteration method since this AMG approach requires lesser device memory than other multigrid methods. The computational burden is distributed between the computational resources using domain decomposition techniques. Besides, we provide a quantitative comparison of the numerical results between using a many-core computing approach and an efficient multi-core implementation. These experimental results show the benefits in performance using multi-GPU systems for addressing large-scale topology optimization problems.

We organize the remainder of the paper as follows. Section 2 introduces the distributed architecture and the required communications system for working the computational resources together. We briefly review the basis and theoretical background of density-based topology optimization in section 3. Section 4 presents the parallel strategy using high-level libraries to effectively take advantage of the computational resources of multi- and many-core architectures. Section 5 shows the numerical experiments evaluating the performance, feasibility, and scalability of the techniques adopted to take advantage of distributed systems using multiple GPUs in density-based topology optimization problems. Finally, section 6 presents the conclusion of the evaluation of the numerical results.

2. Multi-GPU architecture

The use of multiple graphics cards incorporates another level of parallelism to the GPU system, called task-level parallelism. This additional level of parallelism allows us to distribute the workload between the GPUs in the distributed system. Then, we can take advantage of the data-parallelism for which these massive parallel architectures are designed. The use of task-level parallelism allows us to increase flexibility in the implementation, whereas the exploitation of data-level parallelism allows us to scale-up the performance of the application. Figure 1 shows the typical configuration of a GPU cluster, where the computing nodes incorporate many GPUs, namely multi-GPU systems. We install the many-core devices in motherboards using multiple PCIe slots or PCIe slots expansion boards. We can observe that the computational units use their local memory. This makes it necessary to use an efficient mechanism for sharing data between the computational nodes of the high-performance system.

Figure 1 shows the typical deployment connection of a cluster incorporating many-core architectures as co-processors. The head node is the external interface to the GPU cluster. This node receives all external network connections, processes incoming requests, and assigns works to compute nodes. These compute nodes perform the computation using the available computational resources, including multi- and many-core architectures. The main reasons for using a head node are performance and security issues. We dedicate the head node to handle all incoming traffic and manage the work distribution to the compute nodes. For the latter, we adopt the standard Message Passing Interface (MPI), which is commonly used to build applications that can scale in distributed memory systems.

The communication mechanism can use the PCI-e data bus and the network, including Ethernet or Infiniband. These technologies have different memory bandwidth operating at different transfer rates. The communications also introduce latency when data transfer requires computing several operations, such as the operating system (OS) management and the computation of communication protocols. Remote Direct Memory Access (RDMA) technology provides data exchange between devices without involving the OS and CPU. These devices include GPUs, network interface cards (NICs), and storage adapters. The use of RDMA technology allows us to boost the network and host performance by achieving lower latency, lower CPU load, and higher bandwidth. AMD company supports these functionalities using ROCm. We have to consider these facts in the numerical experiments.

3. Density-based topology optimization

Topology optimization consists of solving a binary programming problem that aims to find the optimal material layout minimizing an objective function subject to constraints in the design domain. Density-based topology optimization methods relax the integer-based problem by introducing an interpolation scheme that penalizes a continuous density variable characterizing composite materials [42]. These continuous composites allow the use of gradient-based solvers in the optimization. The problem can be stated as

$$\begin{aligned} \min_{\rho_e} \quad & f(\boldsymbol{\rho}_e, \mathbf{u}) \\ \text{s. t. :} \quad & \mathbf{K}(\boldsymbol{\rho}_e)\mathbf{u} = \mathbf{f} \\ : \quad & V(\boldsymbol{\rho}_e) \leq V^* \\ : \quad & 0 \leq \rho_e \leq 1,\ \boldsymbol{\rho}_e \in \mathscr{D} \end{aligned} \tag{1}$$

where f is the objective function, $\boldsymbol{\rho}_e$ is the vector of density design variables, $\mathbf{u}$ is the system response, $\mathbf{K}$ is the global stiffness matrix, and $\mathbf{f}$ is the force vector. We denote the design domain by $\mathscr{D}$ and constrain the volume of material $V(\boldsymbol{\rho}_e)$ to be smaller than a prescribed target V^*.

This parameterization leads to designs with large areas of intermediate densities numerically optimal but usually impossible to manufacture. We usually address this problem using implicit penalization techniques driving the optimization design to solid/void configurations. The SIMP method makes use of these relaxing techniques using a power-law interpolation function between void and solid to determine the stiffness matrix of each element $\mathbf{K}_e$ as follows

$$\mathbf{K}_e = \mathbf{K}_{min} + {\rho_e}^p(\mathbf{K}_0 - \mathbf{K}_{min}), \tag{2}$$

where $\mathbf{K}_0$ and $\mathbf{K}_{min} > 0$ are the stiffness matrix of solid and void material respectively, and $p > 1$ is the penalization power. We can select p sufficiently big to penalize intermediate densities. According to Bendsøe and Sigmund [42], p $\geq$ 3 is usually required for ensuring that the Hashin-Shtrikman bounds are not violated.

We can apply the SIMP method to different physics problems. In this work, we define the objective function for the minimization of structural compliance (maximization of stiffness) as follows

$$f = c = \mathbf{f}^{\mathrm{T}}\mathbf{u}, \tag{3}$$

where $\mathbf{f}$ and $\mathbf{u}$ are the global force load and the displacement vector, respectively. Considering the discretized linear state system

$$\mathbf{K}\mathbf{u} = \mathbf{f}, \tag{4}$$

and using the adjoint state method, the sensitivity of (3) with respect to the design variables $\boldsymbol{\rho}_e$ is

$$\mathbf{f}_{\rho_e} = \frac{\partial f}{\partial \boldsymbol{\rho}_e} = -\mathbf{u}^{*\mathrm{T}}\frac{\partial \mathbf{K}}{\partial \boldsymbol{\rho}_e}\mathbf{u} = -\mathbf{u}^{*\mathrm{T}} p {\boldsymbol{\rho}_e}^{p-1}(\mathbf{K}_0 - \mathbf{K}_{min})\mathbf{u}, \tag{5}$$

where $\mathbf{u}^*$ is given by the solution of the adjoint problem

$$\mathbf{K}\mathbf{u}^* = \frac{\partial f}{\partial \mathbf{u}}, \tag{6}$$

where the right hand side is $\partial f/\partial \mathbf{u} = \mathbf{f}$ for the minimization of structural compliance, i.e. the problem is self-adjoint and the solution of (6) is $\mathbf{u}^* = \mathbf{u}$.

We usually have to introduce additional constraints to avoid numerical difficulties and modeling problems, such as mesh-dependency of solutions and checker-board patterns [7], respectively. In this work, we adopt the sensitivity filter, which regularizes the sensitivity field using a convolution integral of the product of design variables and sensitivities. We can define the filtered sensitivity variable implicitly as a solution of a Helmholtz type differential equation with homogeneous Neumann boundary conditions [43]. A salient point of this approach is that we do not need to operate with neighbor cells, which is difficult and computationally expensive for complex geometries and design domains distributed among multiple non-overlapping partitions.

The product of design variables in the design domain $\boldsymbol{\rho}_e \in \mathscr{D}$ and sensitivities is denoted as

$$\mathbf{g} = \boldsymbol{\rho}_e \frac{\partial f}{\partial \boldsymbol{\rho}_e}. \tag{7}$$

We can define the implicit form of the convolution integral for sensitivity filter on the domain $\Omega \subset \mathbb{R}^n$ as the solution of the following Helmholtz differential equation with homogeneous Neumann boundary conditions

$$\begin{aligned} -r^2\nabla^2\widehat{g} + \widehat{g} &= g \qquad \widehat{g} \in \Omega \\ \frac{\partial \widehat{g}}{\partial \mathbf{n}} &= 0 \qquad \widehat{g} \in \partial\Omega \end{aligned} \tag{8}$$

where g is the nodal product of design variables and sensitivities, $\widehat{g}$ is the filtered field, and r is a length parameter playing a similar role as the radius of the convolution integral for calculating the sensitivity filter. We can obtain a relationship between length scales to configure the equivalent r parameter to the corresponding physical radius r_p of the convolution integral. We approximate the projection between nodal and elemental g variables using interpolation functions, and then integrating over the design domain. We can solve this problem efficiently using the

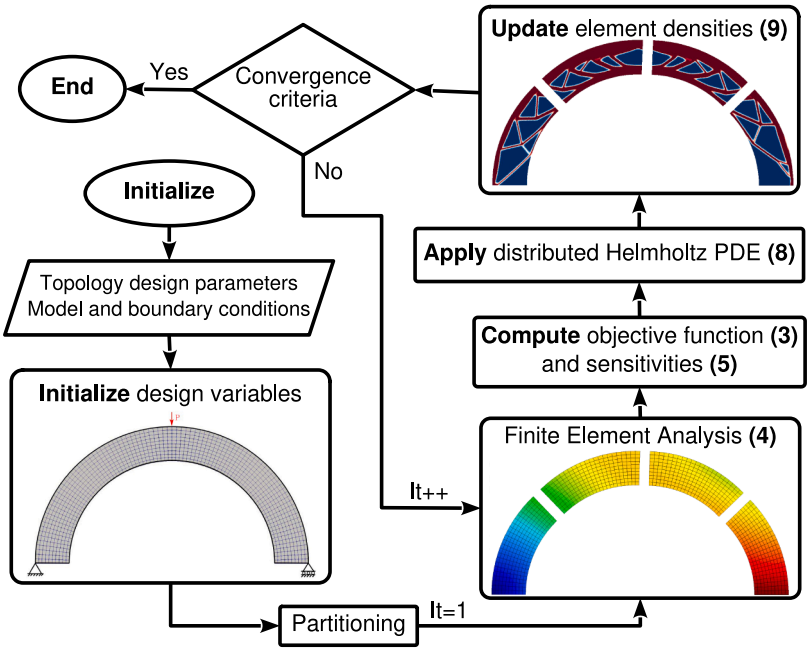


Fig. 2. Flowchart of the distributed density-based topology optimization.

parallel finite element method presented below. After obtaining the nodal values of the filtered field $\widehat{g}$, the filtered sensitivity for a given element is obtained by averaging the nodal values of the filtered field and dividing by the element density [43]. We have to mention that recent alternative formulations are proposed to accommodate design interfaces located along the design domain boundary [44], and thus avoiding the sticking effect.

We update the design parameters using the Optimality Criteria (OC) scheme proposed by Bendsøe [45] for its numerical efficiency. The OC updating scheme is as follows

$$\rho_{e_{k+1}} = \begin{cases} \max\{(1-\zeta)\rho_{e_k}, 0\} & \text{if } \rho_{e_k} B_{e_k}^{\eta} \leq \max\{(1-\zeta)\rho_{e_k}, 0\}, \\ \min\{(1+\zeta)\rho_{e_k}, 1\} & \text{if } \min\{(1+\zeta)\rho_{e_k}, 1\} \leq \rho_{e_k} B_{e_k}^{\eta}, \\ \left(\rho_{e_k} B_{e_k}^{\eta}\right)^{q} & \text{otherwise,} \end{cases} \tag{9}$$

where ζ is a positive step width, η is a numerical damping coefficient, q is a penalty factor to further achieve black-and-white topologies (typically $q = 2$) and B_{e_k} is found from the optimality condition as

$$B_{e_k} = -\frac{\partial f(\boldsymbol{\rho}_e)}{\partial \rho_e}\left(\lambda \frac{\partial V(\boldsymbol{\rho}_e)}{\partial \rho_e}\right)^{-1}, \tag{10}$$

where the Lagrange multiplier λ is found using a bisection algorithm. The algorithm stops when we reach the maximum number of iterations or when the change variable $\| \rho_{e_{k+1}} - \rho_{e_k} \|_{\infty}$ and the change in the objective function $|f_{k+1} - f_k|$ fall below a prescribed value.

4. Parallel strategy

The partition of complex models into smaller and more manageable pieces is a common approach to make use of distributed computational resources. This approach allows us to divide the computational burden and the memory requirements across the computational resources of the distributed computational system. The underline idea consists of partitioning the domain into a set of subdomains. We then solve these subdomains in parallel. This strategy increases the overall performance by distributing computing and memory requirements. We know the techniques using this strategy as Domain Decomposition Methods (DDMs). These methods are iterative in nature and require communication between the computational processes. Their performance depends on the workload balancing of the subdomains and the volume of communications between them. Efficient partitioning techniques allow optimizing the former factor, whereas the variants of DDMs aim to optimize the latter. For simplicity, we adopt a simple Global Subdomain Implementation (GSI) [46], which distributes the operations across the subdomains using communications.

Figure 2 shows the flowchart of the parallel implementation of the distributed density-based topology optimization. We divide the problem into several subdomains to perform the recursive stages of the topology optimization method. In particular, the solving of FEA, the calculation of the objective function and sensitivities, the solving of the distributed sensitivity filter, and the update of design variables. We present below the details of the parallel implementation of such stages.

4.1. Domain partitioning

A key point of density-based topology optimization is that we use the same partitioning during the whole process. Thus, we only have to make the partitioning into several non-overlapping subdomains in the initialization, and we use these subdomains in the stages of the optimization loop shown in Figure 2. The partitioning algorithm takes the resulting mesh of the geometric discretization to divide it following optimization criteria. In particular, the minimization of the number of interface elements reducing the data exchange between processes making the computation associated with each part.

We generate a dual graph from the mesh of the finite element model to perform the partitioning. Each finite element becomes a vertex of the dual graph, and we then use a multilevel k-way partitioning method [47] to define the subdomains considering optimization criteria. These optimization criteria include the minimization of the resulting subdomain connectivity graph and the contiguous partition enforcement. The efficiency of the partitioning is of paramount importance since the partition method is memory-intensive, which is particularly true for large-scale problems and for partitioning with a high number of subdomains. We use ParMETIS [48] library to perform this parallel partitioning using the MPI standard.

4.2. Distributed solving

Let consider the linear system of equations

$$Ax = b, \tag{11}$$

where $A \in \mathbb{R}^{n \times n}$ is the coeffient matrix, $b \in \mathbb{R}^{n \times 1}$ is the right-hand side vector, $x \in \mathbb{R}^{n \times 1}$ is the solution vector, and n is the number of unknowns. We require a distributed representation of the coefficient matrix and vectors to solve the Helmholtz PDE (8) and the linear elasticity (4) problems using multi- and many-core architectures. Let us assume that the coefficient matrix A using compressed sparse row (CSR) format is distributed across $p = \{1, \cdots, n_p\}$ processes (with n_p the number of processes) by contiguous blocks of rows as follows

$$A = \begin{pmatrix} A^0 \\ \vdots \\ A^{p-1} \end{pmatrix}, \tag{12}$$

where the computation of each block submatrix $A^{p-1} \in \mathbb{R}^{n^{p-1} \times n}$ is performed by one single processor p with n^{p-1} the number of rows of the block submatrix and n the number of columns of A. The block submatrices use the global row indices $\{\mathbf{n^0}, \cdots, \mathbf{n^{p-1}}\}$ to facilitate the operations between subdomains. We store these submatrices into local A_{loc}^{p-1} and remote A_{rem}^{p-1} parts. The former is a square matrix with the "local" n^{p-1} unknowns, whereas the latter is a matrix containing coefficients with global column indices stored in other processors. This data structure allows us to differentiate between "local" and "distributed" computation. Local computation is performed with the data stored in the own process p, whereas distributed computation requires some communication mechanism. We adopt a similar approach for distributed dense vectors $b \in \mathbb{R}^{n^{p-1}}$ using the global row indices $\{\mathbf{n^0}, \cdots, \mathbf{n^{p-1}}\}$.

The communications make use of the standardized and portable MPI mechanism. Each process p calculates in the initialization the global columns required from both the own and other processes. Data exchange consists of receiving ghost values from the processes sharing unknowns (global column indices) and then sending the data to the processes that require them to form the gather vectors. We perform data exchange directly between processes since each processor p knows its receive and send processors. This data exchange procedure reduces the computational complexity and the storage requirements because the number of neighbors and the amount of data are independent of the number of p processors.

We address the linear system of equations arising from PDE (8) and (4) using a distributed conjugate gradient iterative solver preconditioned with an aggregation-based AMG method. The use of multigrid methods as a preconditioner of Krylov subspace methods is very popular in the context of structural mechanics. The underline idea is that the interpolation operator of the multigrid method will hardly be optimal, which makes it less efficient for some specific error components. The convergence is slow when this occurs despite almost all of the error components are reduced quickly. The use of Krylov subspace methods usually eliminates these error components efficiently. The use of some iterative solver is a more efficient solution than improving the

Input: $\mathbf{A}$, $\mathbf{B}$, n
Output: $\mathbf{A_1}, \ldots, \mathbf{A_{n_L}}$, $\mathbf{P_0}, \ldots, \mathbf{P_{n_L-1}}$, $\mathbf{R_0}, \ldots, \mathbf{R_{n_L-1}}$

$\mathbf{A_0} \leftarrow \mathbf{A}$, $\mathbf{B_0} \leftarrow \mathbf{B}$, $n_0 \leftarrow n$, $l \leftarrow \emptyset$, $L \leftarrow l$;
while n_l is too big to solve **do**
  $\mathbf{C_l} \leftarrow strength(\mathbf{A_l})$; // Strength-of-connections
  $\mathbf{Agg_l} \leftarrow aggregate(\mathbf{C_l})$ // Construct coarse aggregates
  $\tilde{\mathbf{P}}_\mathbf{l}, \mathbf{B_{l+1}} \leftarrow prolongate(\mathbf{Agg_l}, \mathbf{B_l})$; // Tentative
  $\mathbf{P_l} \leftarrow \tilde{\mathbf{S}}_\mathbf{l}\tilde{\mathbf{P}}_\mathbf{l}$; // Improve tentative interpolation
  $\mathbf{R_l} \leftarrow \mathbf{P_l^T}$;
  $\mathbf{A_{l+1}} \leftarrow \mathbf{R_l A_l P_l}$; // Galerkin projection
  $l \leftarrow l + 1$, $L \leftarrow l$, $n_0 \leftarrow n$;
end

Algorithm 1. AMG setup

Input: $\mathbf{r_l}$, $\mathbf{s_l}$, l
Output: $\mathbf{s_l}$

// Pre-smooth/-relaxation: smooth μ_1 times on $\mathbf{A_l}$ $\mathbf{s_l} = \mathbf{r_l}$
$\mathbf{s_l} \leftarrow \text{Smooth}(\mathbf{r_l}, \mathbf{A_l}, \mathbf{s_l}, \mu_1)$;
$\mathbf{r_l} \leftarrow \mathbf{r_l}$ - $\mathbf{A_l}$ $\mathbf{s_l}$; // Compute residual
$\mathbf{r_{l+1}} \leftarrow R_l$ $\mathbf{r_l}$; // Restict residual to coarse grid
// Coarsest (last level L) grid
if *(l+1 == L)* **then**
  // Direct solver on CPU
  $\mathbf{s_{l+1}} \leftarrow \mathbf{A_{l+1}}^{-1}\mathbf{r_{l+1}}$;
else
  $\mathbf{s_{l+1}} \leftarrow \text{V-cycle}(\mathbf{r_{l+1}}, 0, \text{l+1})$; // Recursion
  $\mathbf{s_l} \leftarrow \mathbf{s_l} + P_l$ $\mathbf{s_{l+1}}$; // Prolongation
  // Post-smooth/-relaxation: smooth μ_2 times on $\mathbf{A_l}$ $\mathbf{s_l} = \mathbf{r_l}$
  $\mathbf{s_l} \leftarrow \text{Smooth}(\mathbf{r_l}, \mathbf{A_l}, \mathbf{s_l}, \mu_2)$;
end

Algorithm 2. AMG preconditioner (V-cycle)

Input: $\max_{\text{iter}}$, tol, tol_{abs}, $\mathbf{A}$, $\mathbf{f}$, $\mathbf{u_0}$
Output: $\mathbf{u}$, $||\mathbf{r}||_2 / f_{norm}$, it

$it \leftarrow \emptyset$, $\gamma \leftarrow \emptyset$, $\mathbf{s} \leftarrow \emptyset$, $\mathbf{p} \leftarrow \emptyset$, $\mathbf{q} \leftarrow \emptyset$, $\mathbf{u} \leftarrow \mathbf{u_0}$;
$f_{norm} \leftarrow ||\mathbf{f}||_2$;
$\mathbf{r} \leftarrow \mathbf{f}$ - $\mathbf{A}\,\mathbf{u}$;
$\varepsilon \leftarrow \max(\text{tol} \cdot ||\mathbf{r}||_2, \text{tol}_{\text{abs}})$;
while $(it < max_{iter})$ && $(||\mathbf{r}||_2 > \varepsilon)$ **do**
  $\mathbf{s} \leftarrow$ V-cycle$(\mathbf{r}, \mathbf{s}, \emptyset)$; // AMG V-cycle to estimate $\mathbf{A}^{-1}\mathbf{r}$
  $\gamma_{\text{old}} \leftarrow \gamma$, $\gamma \leftarrow \mathbf{r}^T\mathbf{s}$;
  if $(it == \emptyset)$ **then**
    $\mathbf{s} \leftarrow \mathbf{p}$;
  else
    $\mathbf{p} \leftarrow \mathbf{s} + \gamma/\gamma_{\text{old}}\mathbf{p}$; // axpby operation
  end
  $\mathbf{q} \leftarrow \mathbf{A}\,\mathbf{p}$; // spmv operation
  $\alpha \leftarrow \gamma/\mathbf{p}^T\mathbf{q}$;
  $\mathbf{u} \leftarrow \mathbf{u} + \alpha\,\mathbf{p}$; // axpby operation
  $\mathbf{r} \leftarrow \mathbf{r}$ - $\alpha\,\mathbf{q}$; // axpby operation
  $it \leftarrow it + 1$;
end

Algorithm 3. Conjugate gradient algorithm

construction of the interpolation operator. We choose aggregation-based AMG methods as a preconditioner because they define the interpolation as a piecewise constant operator with only one non-zero per row. This fact has a particularly beneficial effect on the performance of GPU computing due to the significant reduction of memory requirements in comparison with classical AMG [49] and the performance improvement of the interpolation operator.

Multigrid methods use a two-grid scheme to address the problem. Let l be the grid level and n_l the number of unknowns in that l level. Considering the initialization $A_0 = A$ and $n_0 = n$, we define the coarse grid coefficient matrices A_{l+1} recursively as follows

$$A_{l+1} = R_l A_l P_l, \tag{13}$$

where $P_l \in \mathbb{R}^{n_l \times n_{l+1}}$ is the interpolation or prolongation operator, $R_l \in \mathbb{R}^{n_{l+1} \times n_l}$ (normally obtained as P_l^T) is the restriction operator, and $A_l \in \mathbb{R}^{n_l \times n_l}$ and $A_{l+1} \in \mathbb{R}^{n_{l+1} \times n_{l+1}}$ are the fine and coarse grid coefficient matrices, respectively. We calculate recursively such transfer operators until a grid level L in which the number of unknowns n_L is sufficiently low to solve it in a reasonable time, typically using a direct solver.

Algorithm 1 describes the AMG setup for aggregation-based AMG methods. We calculate the grid hierarchy and the transfer operators for all the levels l using the algebraic properties of the coefficient matrix A. The setup requires the coefficient matrix $A \in \mathbb{R}^{n \times n}$ and the near null-space vector $B \in \mathbb{R}^{n \times m}$. It uses strength, aggregate, tentative, prolongate, and Galerkin projection operations. The strength operator constructs a graph C_l of strong connections at the level l evaluating the relationship between unknowns with the diagonal and outside diagonal coefficients using a coarsening ratio. The aggregation process uses the graph C_l to select independent grid points as root nodes grouping/aggregating them with their neighbors. We build the tentative prolongation matrix $\widetilde{P}_l$ as a simple grid transfer operator by a piecewise constant interpolation. We then apply a smoother $\widetilde{S}_l$ to the tentative prolongation matrix $\widetilde{P}_l$ to obtain a more robust prolongation operator P_l. Finally, we perform the sparse Galerkin product using two matrix-matrix products.

Algorithm 2 shows the pseudo-code of the V-cycle of aggregate AMG for preconditioning the distributed conjugate gradient iterative solver. This algorithm aims to approximate the solution of (11) given the residual of the previous estimation of the iterative solver. The procedure consists of the application of μ_1 smoothing operations to the approximate solution s_l at the level l and the computation of the residual r_l for the relaxed approximate solution s_l. We then restrict the residual to the coarse grid and solve the linear system if we reach the last level L. We prolongate the solution s_l at the coarsest grid to the finer one applying μ_2 smoothing operations to the approximate solution.

Algorithm 3 details the pseudo-code of the distributed conjugate gradient algorithm using the V-cycle of aggregate AMG as a preconditioner. This iterative solver requires the maximum number of iterations $\max_{\text{iter}}$, the tolerance $<$ tol, $\text{tol}_{\text{abs}}>$, the coefficient matrix $\mathbf{A}$, the right-hand side $\mathbf{f}$, and the initial guess for initializing the iterative procedure. The recursive solver provides an approximate solution $\mathbf{u}$ of (11) with a residual after it iterations.

We use high-level open-source libraries to effectively integrate these functionalities in the topology optimization process using the computational resources of multi- and many-core architectures. In particular, we use AMGCL [50] for the distributed Krylov iterative solver and the multigrid preconditioning, and VEXCL [51,52] for the use of multiple GPUs in the development using OpenCL.

4.3. Elemental-based calculation

Once we divide the domain into several non-overlapped subdomains, the parallel computation of the objective function (3), the sensitivities (5), and the update of design variables (9) is straightforward because these operations only require the information of the own finite elements.

Fig. 3. Distributed system with two compute nodes incorporating four AMD Radeon VII each one.

Thus, the computation needed by each subdomain is performed by one single processor or co-processor for multi- and many-core computation, respectively. We use the VEXCL open-source library to facilitate the development using OpenCL.

5. Numerical Experiments

We evaluate the benefits and limitations of using a distributed system with multiple GPUs for addressing the density-based topology optimization problem. It is well-known that FEA is the principal bottleneck of the topology optimization pipeline, and thus the experiments focus on the evaluation of the use of many-core devices to address the solving stage during the topology optimization process. We aim to evaluate the scalability of these techniques analyzing the pros and cons, and comparing the results with efficient multi-core systems using the same strategy. The numerical experiments consist of solving different three-dimensional problems with different discretization sizes using structured and unstructured meshes with diverse types of finite elements.

We run the numerical experiments using a set of workstations working as a computer cluster, as shown in Figure 1. We use four compute nodes for the numerical experiments with an Intel Xeon W-2145 CPU at 3.70GHz and 96GB of RAM. A couple of the compute nodes incorporates four low-cost AMD Radeon VII graphics cards. We use these compute nodes to evaluate the performance of the distributed system using GPU computing. The AMD Radeon VII graphics card incorporates the Vega 20 processor built on the 7 nm process with 13.23e9 transistors. It features 3840 shading units, 240 texture mapping units, and 64 ROPs. The 3840 stream processors operate at a frequency from 1400 MHz (base) to 1750 MHz (boost). It has 16GB of HBM2 memory running at 1000 MHz and a memory interface of 4096-bits, which provides a total memory bandwidth of around 1024GBps. A key of this graphics card is that it includes specific hardware to perform double-precision floating-point operations. The theoretical performance is 13.44 TFLOPS and 3.36 TFLOPS for single- and double-precision floating-point operations, respectively. The significant double-precision computational capability (rate 1:4 w.r.t. single-precision) is only included on the graphics cards designed for scientific computation purposes. The thermal design power (TDP) of AMD Radeon VII is around 295W, which allows us to install four graphics devices using a high power PSU. Figure 3 shows the two compute nodes incorporating a multi-GPU system with four AMD Radeon VII each node. We connect the head and compute nodes using a 10 Gigabit Ethernet network.

The hardware allows us to run all the experiments using both multi- and many-core architectures using different computational resources to evaluate the scalability. The multi-core numerical experiments can use up to 32 cores in four compute nodes using distributed memory, whereas many-core numerical experiments can use eight GPU devices using two compute nodes. We compile the source code of the numerical experiments using the AMDGPU driver version 19.50.

Table 1
Geometric and topology design parameters for the experiments.

Geometry (meters)					Topology parameters		
Cantilever beam							
		L	**W**	**H**	**v (%)**	**p**	r_p (m)
		2.0	1.0	1.0	12	3.0	0.08
Wheel rim							
L	**H**	$\mathbf{D}_1$	$\mathbf{D}_2$	$\mathbf{D}_3$	**v (%)**	**p**	r_p (m)
0.155	0.1	0.108	0.424	0.4318	50	3.0	0.04
Hollow circular shaft with varying cross-section							
	L	**H**	$\mathbf{D}_1$	$\mathbf{D}_2$	**v (%)**	**p**	r_p (m)
	4	0.9	1	2	50	3.0	0.08
Hollow circular shaft							
		L	**D**	**H**	**v (%)**	**p**	r_p (m)
		4	1	0.9	40	3.0	0.08

The battery of experiments consists of four topology optimization problems using structured and unstructured meshes with different finite elements to show that the approach is not limited to any constraint. In particular, the experiments consist of a cantilever beam, a wheel rim, and a hollow circular shaft with and without varying cross-sections. The first three experiments use a structured mesh, whereas the last one uses an unstructured mesh. Note that the only test using finite elements with a similar aspect ratio is the first one. GPU computing strategies usually uses stencil computation to exploit data locality operating on a regular Cartesian structured mesh. This approach, combined with matrix-free implementations to save device memory, shows good GPU performance. However, this proposal is not adopting stencil computation. We parameterize the mesh with the number of divisions of geometric primitives to facilitate reproducible results. We also increase the problem size by uniformly refining the mesh. Table 1 specifies the geometric and topology optimization parameters of the experiments. We evaluate the effect of the tolerance of the distributed conjugate gradient method presented in Algorithm (3) on the performance. In particular, we use tolerances $tol = \{10^{-6}, 10^{-8}\}$ to test the evolution of optimization and computational performance. We set the stiffness of solid $K_0 = 1$ and void $K_{min} = 10^{-9}$ material in the power-law interpolation function of (2). We also define the set of optimizable and non-optimizable finite elements in the initialization of the topology optimization for each experiment.

We solve the experiments using multi- and many-core computing and compare the GPU instance to an efficient multi-core implementation. We

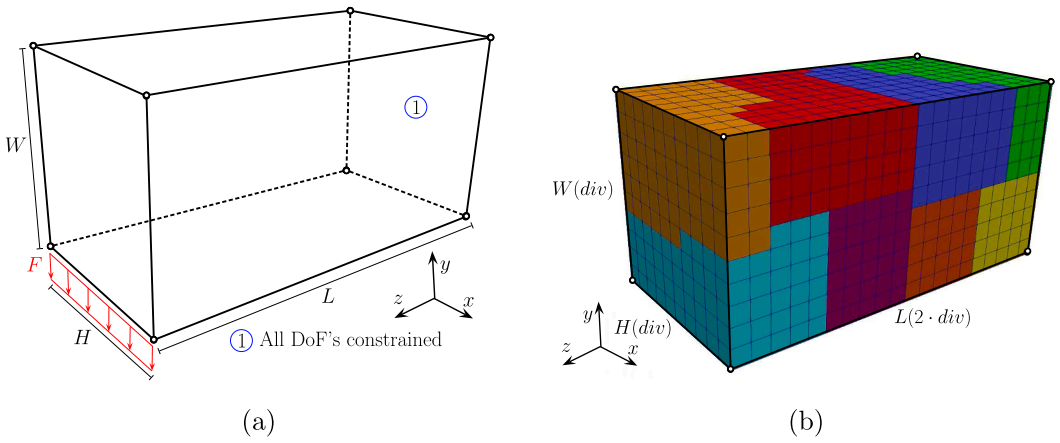


Fig. 4. Cantilever experiment: (a) geometric configuration and boundary conditions, and (b) mesh parameterization and partitioning into eight subdomains.

also evaluate the use of mixed-precision in the FEA solving stage (4), which would be rewarding using GPU computing considering the theoretical computational performance rate 1:4 between single- and double-precision floating-point operations of AMD Radeon VII. The use of mixed-precision consists of using single-precision for the computation of the preconditioning (AMG setup and V-cycles of the aggregate AMG method) and double-precision for the calculations of the iterative solver to avoid convergence problems due to round-off errors. We report the wall-clock time and speedup with the reference of the multi-core implementation for all the numerical experiments. The numerical results provide the information to highlight the benefits and limitations of using many-core architectures in density-based topology optimization. We present below the numerical results of the four density-based topology optimization experiments evaluating the computational aspects of the parallel strategy using multi- and many-core computation.

5.1. Cantilever

The first experiment consists of the topology optimization of a three-dimensional cantilever beam with fixed displacements on one side and loads uniformly distributed along the lower edge of the other side. Figure 4(a) shows the geometry and boundary conditions of the finite element model, indicating the parameters specified in Table 1. This table also includes the topology optimization design parameters. In particular, the volume fraction v, the penalization p, and the physical radius r_p of the convolution integral for calculating the sensitivity filter, which allows us to obtain the variable r of (8) by the relation between length scales [43]. Figure 4(b) depicts a coarse mesh partitioned into eight subdomains using the ParMetis library. We can observe that the partitioning is performed into balanced subdomains enforcing the contiguous partitions. It also shows the parameterization of the tessellation with the *div* variable for the geometric variables parameters specified in Table 1. We use such a *div* parameter together with the refinement of l_{ref} levels to obtain the mesh with the number of design variables to optimize.

We tessellate the cantilever using a structured mesh composed of eight-node linear hexahedral elements. One refinement level of a mesh with hexahedral elements refines the mesh dividing each finite element into eight new bricks. There are several choices for the smoothing iterations or relaxation operator mentioned in Algorithm 2. We choose the incomplete LU factorization with zero fill-ins (ILU0) for the multi-core solving and several Jacobi iterations for the many-core solving to prevent using device memory-consuming algorithms, which seriously affect the GPU performance. We initialize the topology optimization with a similar density design for all the finite elements satisfying the volume constraint. All the finite elements are optimizable for this experiment.

We evaluate the strong scaling of the GPU implementation using a problem with memory requirements that fit on the device memory of one GPU. We obtain this model with two million finite elements (6151203 unknowns) configuring the parameter $div = 100$ without any refinement of the mesh. Figure 5(a) shows the strong scaling of many-core implementation using a different number of GPU devices and computing nodes with tolerance $tol = 10^{-8}$ for the FEA stage. We also evaluate the performance improvement reducing the tolerance ($tol = 10^{-6}$) and reusing the displacement solution of the previous iteration, showing the results in Figure 5(b). These experiments detail the wall-clock time for the FEA solving stage (4) during the 204 iterations of the topology optimization process. We group the experiments into two sets: topology optimization using one (continuous line) and two (dashed line) compute nodes. The distributed solving strategy makes use of mixed-precision for all the topology optimizations. We can observe that the reduction of the tolerance of the iterative solver reusing the solution of the previous iteration can improve the performance significantly, especially when the topology of the last iteration is close to the current design. We also can observe that the wall-clock time reduces as increasing the number of GPU devices for solving. However, the performance improvement decreases as the problem size is smaller in each subdomain. This fact is also due to communications between devices and nodes required by the GSI approach. We can observe that the performance improvement using the third device is higher than using the second device. We attribute this fact to the intra-node communications needed by the second device in comparison with all the computations performed by only one device. This problem is exacerbated, together with the problem size of each subdomain, when we require inter-node communications.

We also evaluate the weak scaling of the GPU instance by solving topology optimization problems with a size of one million finite elements (3168963 unknowns) per GPU device. We generate the models adjusting the *div* and l_{ref} parameters. Figure 6 shows the weak scaling tests of many-core implementation with a tolerance $tol = 10^{-6}$ for solving FEA reusing the displacement solution of the last topology optimization iteration. These experiments detail the wall-clock time for the FEA solving (4) using mixed-precision during the 204 iterations of the topology optimization process. We can observe that the performance of the GPU instance initially degrades as we use new devices to solve the model with one million finite elements per device. As in the previous experiments, we observe that this effect exacerbates when we incorporate the second GPU device for the resolution, i.e., once we start to use intra-node communications. The strong scaling shows better performance as we include new GPU devices. We attribute this effect to efficient partitioning minimizing the connections between the subdomains. We should remark that the communication mechanism exchanges data directly between processes as detailed in section 4, and thus the scalability improves as the number of connections between GPU devices does not increase with the problem size.

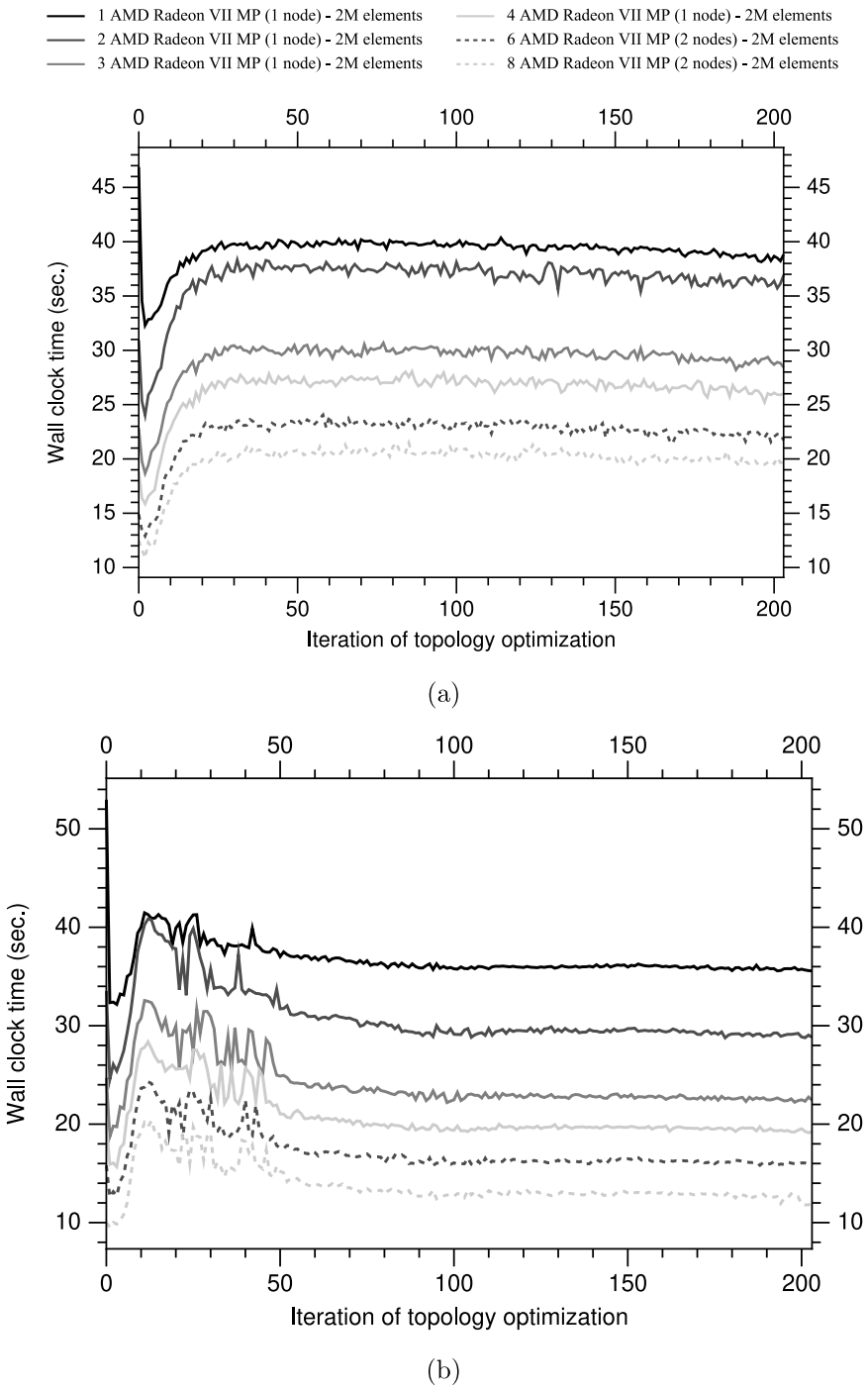


Fig. 5. Strong scaling for solving FEA of the cantilever experiment using two million hexahedral finite elements (6151203 unknowns): with tolerance (a) 10^{-8} and (b) 10^{-6} and reusing the solution of the last iteration.

Figure 7 shows the wall-clock time of AMG stages for the weak scaling evaluation of the GPU instance. We use one million finite elements (3168963 unknowns) per GPU device and a tolerance $tol = 10^{-8}$ for the distributed conjugate gradient. The AMG stages are the AMG setup and solving step. The former aims to find the proper transfer operators using the algebraic properties of the coefficient matrix. This objective is a challenging task, and the algorithms used to find these transfer operators are intrinsically serial. The latter computes the iterations of the distributed conjugate gradient algorithm. It uses the V-cycle of aggregate AMG as a preconditioner. Once we have the hierarchy, the solving is easily parallelizable, usually using sparse matrix-vector products. Figure 7(a) shows the AMG setup timing and the total wall-clock time for solving. We can observe that AMG setup consumes a significant fraction of the time for solving. Figure 7(b) shows the wall-clock time per conjugate gradient iteration, including the V-cycle of aggregate AMG as a preconditioner. We can observe that the time required to perform one conjugate gradient iteration is low. The ratio between both stages depends on the converge of the distributed conjugate gradient algorithm. We have to remark that using parallel computing can be more beneficial to reduce the effort in the AMG setup performing a high number of conjugate gradient iterations in parallel.

We finally evaluate the performance of solving FEA with a higher number of unknowns than the previous tests. We configure the parameter $div = 10$ and the number of refinement levels $l_{ref} = 4$ to obtain a

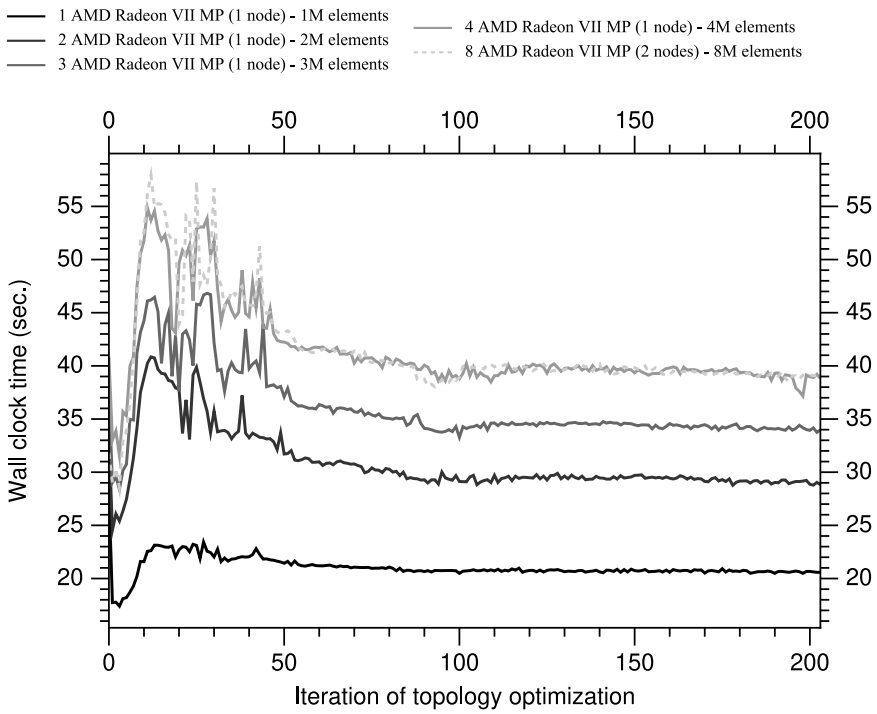


Fig. 6. Weak scaling for solving FEA of the cantilever experiment with one million finite elements (3168963 unknowns) per GPU device and tolerance 10^{-6} reusing the solution of the last iteration.

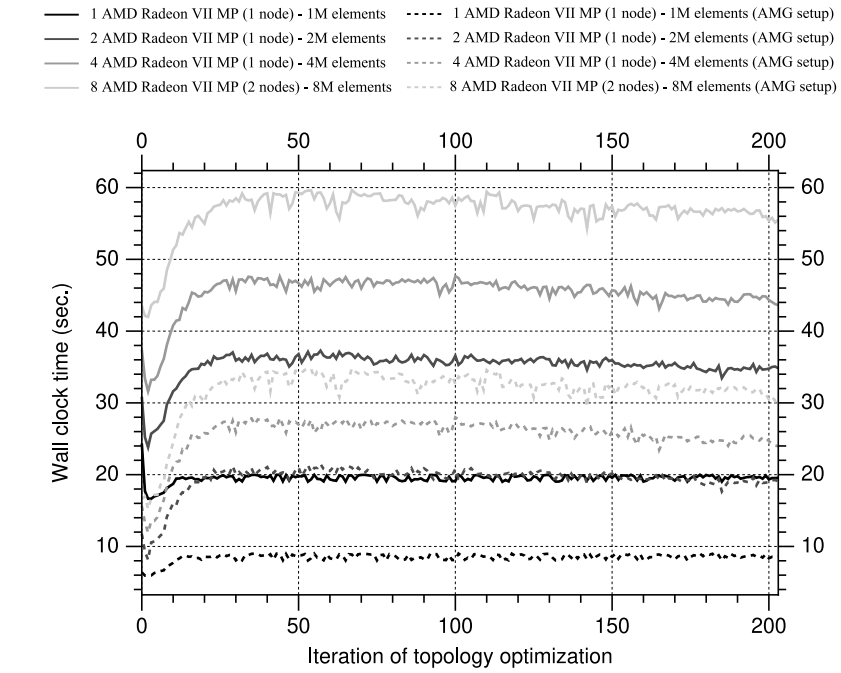
finite element model with 8192000 hexahedral finite elements (24961923 unknowns). Figure 8(a) shows the wall-clock time for solving FEA during the optimization using a tolerance $tol = 10^{-8}$ for the iterative solver. Figure 8(b) shows similar information about the same experiment but using a tolerance $tol = 10^{-6}$ for the iterative solver and initializing it with the result of the last iteration of topology optimization. These experiments allow us to evaluate the performance and feasibility of this approach. We perform all the optimizations with multi- and many-core computing using mixed- and double-precision. The multi-core experiment makes use of 32 cores using four compute nodes, whereas the many-core experiment uses 8 GPU devices using two compute nodes. We observe that the reduction of the tolerance of the iterative solver and reusing the solution of the previous iteration improve the performance meaningfully, especially when the topology is close to the previous design. Figure 9 shows the evolution of the objective function during the topology optimization. We obtain similar convergence of the topology optimization objective function by reducing the tolerance of the iterative solver. We can observe that solving using mixed-precision is rewarding both it using multi- and many-core computing.

We also show the speedup between multi- and many-core computing using mixed- and double-precision. We obtain a speedup with the experiments using a tolerance $tol = 10^{-8}$ for the iterative solver of around 3.2x and 2.8x using mixed- and double-precision, respectively. We observe a moderate reduction of the speedup reducing the tolerance of the iterative solver. We attribute this fact to the fewer iterations needed by the distributed iterative conjugate gradient solver, where the V-cycles of the AMG method show higher speedups than the AMG setup. We obtain higher acceleration using mixed precision with GPU computing. However, the increase in performance is lower than the theoretical rate of 1:4 between single- and double-precision floating-point operations of AMD Radeon VII. We should remark that only the preconditioning uses single-precision floating-point computations and that the computing is only a part of the wall-clock time since communications are needed. In any case, we can obtain higher performance improvement using graphics devices with lower computational performance rates.

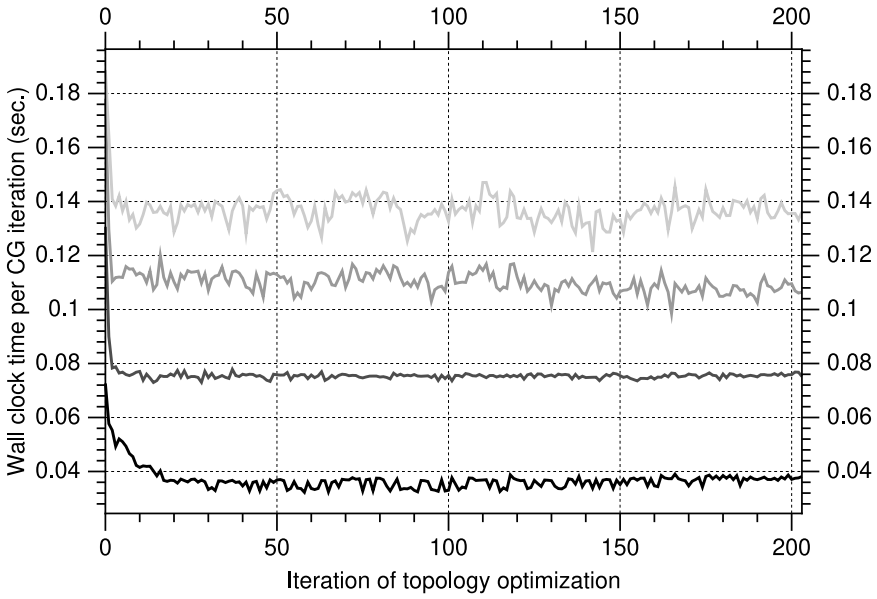
We can find this experiment in [20], being available as the default example in the open-source PETSc based topology optimization project associated with such a publication. This project uses the DMDA functionalities of PETSc [53], which provide us an interface for both the topology and geometry of a Cartesian structured mesh. DMDA also includes the capability of parallel refinement and coarsening. This library makes use of stencil computation to update data points of the Cartesian structured mesh according to some fixed pattern, called a stencil. The PETSc based code provides good performance combining GMRES iterative solver preconditioned with a geometric multigrid method and stencil computation. We have compared the presented multi-core implementation with the PETSc based code obtaining wall-clock times of the same order of magnitude configuring similar tolerances for the iterative solvers. Besides, the use of aggregation-based AMG requires lower levels than classical AMG [54], and thus the memory requirements of the proposed implementation are significantly lower than the PETSc based code. The presented implementation allows us to run the previous topology optimization using only two computing nodes, as shown in the GPU computing results. We have to remark that we can obtain reduced speedup using GPU computing if the problem size is not high enough to take advantage of the potential of many-core devices, as we show above in the strong scaling evaluation of the GPU implementation.

5.2. Wheel rim

The second experiment consists of the topology optimization of a wheel rim with fixed displacements in the center bore and six loads uniformly distributed along with the line segments in the wheel rim diameter dividing it into six equal parts. Figure 10(a) shows the geometry and boundary conditions of the finite element model, indicating the parameters specified in Table 1. In particular, the wheel rim width L, the wheel bore H, and the wheel rim diameter D_3. The values of these parameters correspond to a wheel rim for 155/90R17 tires. We also specify two intermediate diameters (D_1 and D_2) to define the non-optimizable volumes of the design for the connection with the axle and with the



(a)



(b)

Fig. 7. Wall-clock time for the solving of the cantilever experiment with one million finite elements (3168963 unknowns) per GPU device and tolerance 10^{-8}: (a) total wall-clock time and AMG setup timing, and (b) wall-clock time per conjugate gradient iteration.

tire, as shown in Figure 10(c). Figure 10(b) depicts a coarse mesh partitioned into six balanced subdomains enforcing the contiguous partitioning. It also shows the tessellation of the geometric parameters specified in Table 1 using the div, div_r, and div_c parameters. We use these parameters, together with the refinement of l_{ref} levels, to obtain the mesh with the number of design parameters to optimize.

We tessellate the wheel rim problem using a structured mesh composed of four-node linear tetrahedral elements. One refinement level of a mesh with tetrahedral elements refines the mesh dividing each finite element into eight new tetrahedra. We choose the simplest and parallel smoother sparse approximate inverse (SPAI-0) for the relaxing iterations of Algorithm 2. We initialize the non-optimizable finite elements with the maximum design variable $\rho_e = 1$, whereas the optimizable finite elements are initialized with similar density design variables satisfying the volume constraint.

We evaluate the performance and scalability of solving FEA using mixed- and double-precision during the topology optimization process. We also evaluate the performance and feasibility of reducing the tolerance of the iterative solver and reusing the displacement result of the previous iteration. We obtain a finite element model with 88473600 tetrahedra finite elements (45023040 unknowns) configuring the parameters $div = 30$, $div_r = 26$, $div_c = 1$, and the number of refinement levels $l_{ref} = 3$. Figure 11(a) shows the wall-clock time for solving FEA (4)

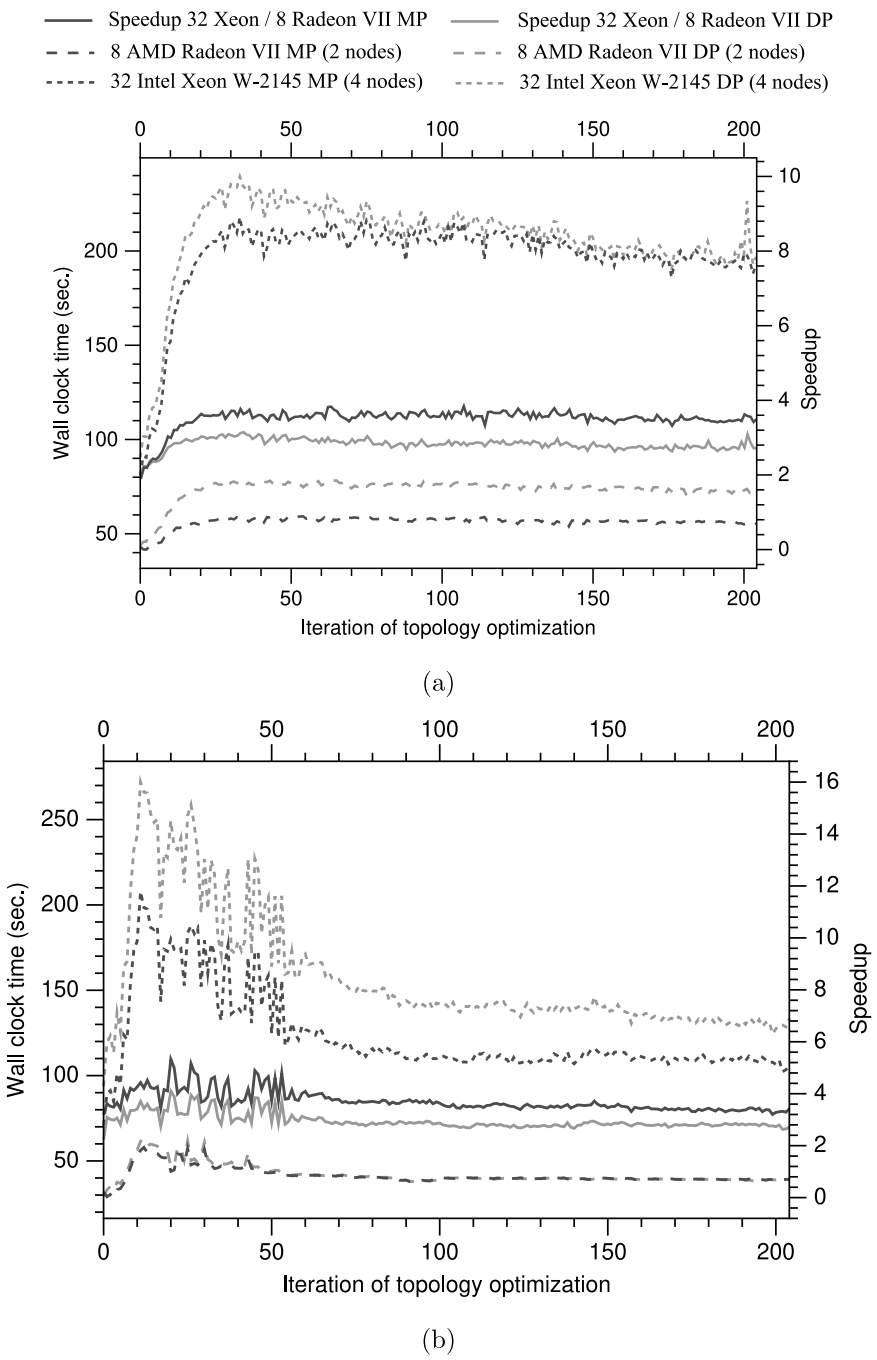


Fig. 8. Wall-clock time for solving FEA of the cantilever experiment using eight million hexahedral finite elements (24961923 unknowns): with tolerance (a) 10^{-8} and (b) 10^{-6} reusing the solution of the previous iteration.

using a tolerance $tol = 10^{-8}$ for the iterative solver during the 58 iterations of the topology optimization process using multi- and many-core computing. Figure 11(b) shows the wall-clock time for solving FEA using a tolerance $tol = 10^{-6}$ and reusing the result of the displacement field of the last optimization iteration as the initial seed of the iterative solver. We obtain a speedup of around 4.8x and 3.9x using mixed- and double-precision, respectively. As in the previous experiment, we observe a moderate reduction of the speedup reducing the tolerance of the iterative solver. Nevertheless, we reduce the wall-clock time for solving FEA meaningfully, especially when the topology is similar to the obtained in the previous iteration of the optimization process. We attribute this fact to the significant reduction of the iterations of the distributed iterative solver. We have checked that the evolution of the objective function during the topology optimization is similar to reducing the tolerance of the iterative solver, showing Figure 12 such an evolution and the final design. We also have to mention that the speedups are significantly higher than the obtained using a structured hexahedral mesh with the ILU0 relaxation operator.

5.3. Hollow circular shaft with varying cross-section

The third experiment consists of the topology optimization of a three-dimensional hollow circular shaft with a varying cross-section with fixed displacements on one side and four loads uniformly distributed along the line segments dividing the other side into four equal parts. Figure 13(a) shows the geometry and boundary conditions of the

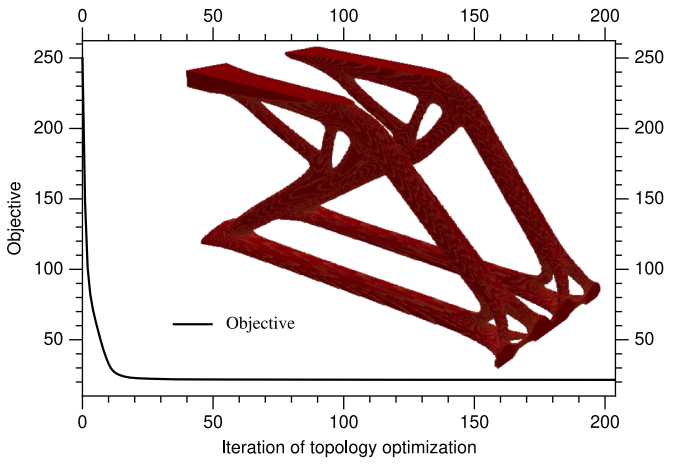


Fig. 9. Evolution of the objective function during the topology optimization of the cantilever experiment using eight million hexahedral finite elements (24961923 unknowns).

finite element model, indicating the parameters specified in Table 1; in particular, the length of the shaft L, the diameter D_1 in the sides, the hole diameter H in the sides, and the diameter D_2 in the center of the shaft. We define the cross-section variation using a spline with the points indicated in Figure 13(a). Figure 13(b) depicts a coarse mesh partitioned into six balanced subdomains enforcing the contiguous partitioning. It also shows the tessellation of the geometric parameters specified in Table 1 using the *div* parameter. We use this parameter together with the refinement of l_{ref} levels to obtain the mesh with the number of design parameters to optimize. Figure 13(c) shows the definition of the optimizable and non-optimizable volumes.

We tessellate the hollow circular shaft with varying cross-section experiments using a structured mesh composed of eight-node linear hexahedral elements with a different aspect ratio to fit the complex geometry. We choose the simplest and parallel smoother sparse approximate inverse (SPAI-0) for the relaxing iterations of Algorithm 2. We initialize the non-optimizable finite elements with the maximum design variable $\rho_e = 1$, whereas we initialize the optimizable finite elements with similar density design variables satisfying the volume constraint. We obtain a finite element model with 15630336 hexahedral finite elements (49860864 unknowns) configuring the parameters $div = 10$ and the number of refinement levels $l_{ref} = 3$. Figure 14(a) shows the wall-clock time for solving FEA (4) during the 199 iterations of the topology optimization process using multi- and many-core computing and configuring a tolerance $tol = 10^{-8}$ for the distributed conjugate gradient solver. Figure 14(b) shows the wall-clock time for solving FEA using a tolerance $tol = 10^{-6}$ and reusing the result of the displacement field of the last optimization iteration as the initial seed of the iterative solver. We run all the experiments using mixed-precision. This experiment aims to evaluate the performance of the approach using a non-regular grid. Cartesian regular grids are usually adopted to exploit GPU computing using stencil computation in topology optimization. We run the experiment using 32 cores with multi-core computing and 8 AMD Radeon VII with many-core computing obtaining a speedup of 7.8x using mixed-precision. In this case, we observe a drastic reduction of the speedup reducing the tolerance of the iterative solver. However, we significantly reduce the wall-clock time for solving FEA with the timing until seven times lower in the multi-core implementation. Figure 15 shows the final design and the evolution of the objective function during the topology optimization, which is the same using both tolerances.

5.4. Hollow circular shaft

The last experiment consists of the topology optimization of a three-dimensional hollow circular shaft with fixed displacements on one side and four loads uniformly distributed along the line segments dividing the other side into four equal parts. Figure 16(a) shows the geometry and boundary conditions of the finite element model, indicating the parameters specified in Table 1. In particular, it shows the length L, the diameter D, and hole diameter H of the shaft. Figure 16(b) depicts a coarse mesh partitioned into six balanced subdomains enforcing the contiguous partitioning. It also shows the tessellation of the geometric parameters specified in Table 1 using the *div* parameter. We use this parameter, together with the refinement of l_{ref} levels, to obtain the mesh with the number of design parameters to optimize. Figure 16(c) shows the definition of the optimizable and non-optimizable volumes.

We tessellate the hollow circular shaft problem using an unstructured mesh composed of four-node linear tetrahedral elements. We choose the SPAI-0 relaxing operator for the smoothing iterations of Algorithm 2. We initialize the non-optimizable finite elements with the maximum design variable $\rho_e = 1$, whereas we initialize the optimizable finite elements with similar density design variables satisfying the volume constraint. We obtain a finite element model with 106348544 tetrahedral finite elements (56513256 unknowns) configuring the parameters $div = 20$ and the number of refinement levels $l_{ref} = 4$. This experiment is significantly higher than the previous ones aiming to evaluate the feasibility and performance of the distributed approach to address large-scale problems using low-cost computational resources. Figure 17(a) shows the wall-clock time for solving FEA (4) using a tolerance $tol = 10^{-8}$ for the iterative solver during the 414 iterations of the topology optimization. Figure 17(b) shows the wall-clock time for

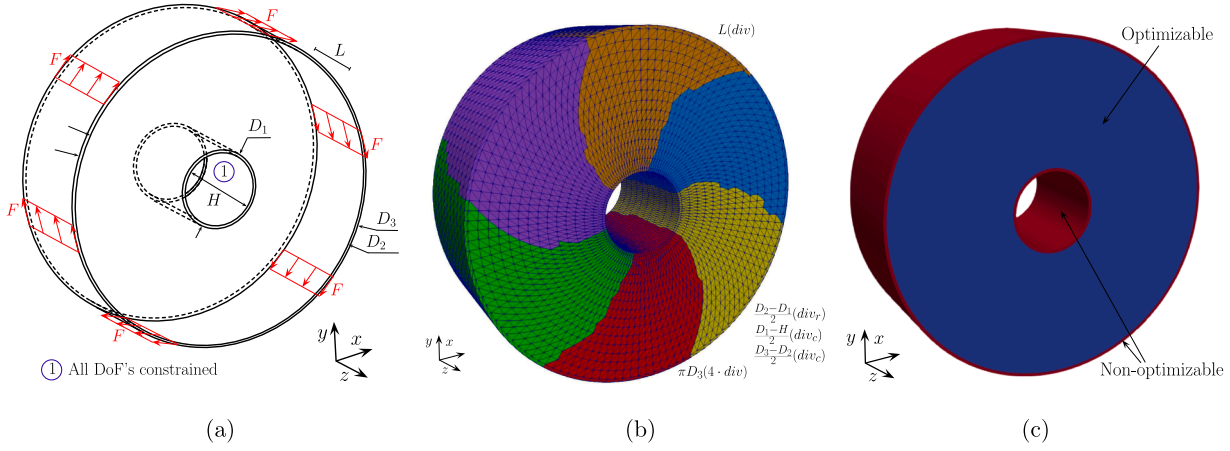


Fig. 10. Wheel rim experiment: (a) geometric configuration and boundary conditions, (b) mesh parameterization and partitioning into six subdomains, and (c) definition of optimizable and non-optimizable volumes.

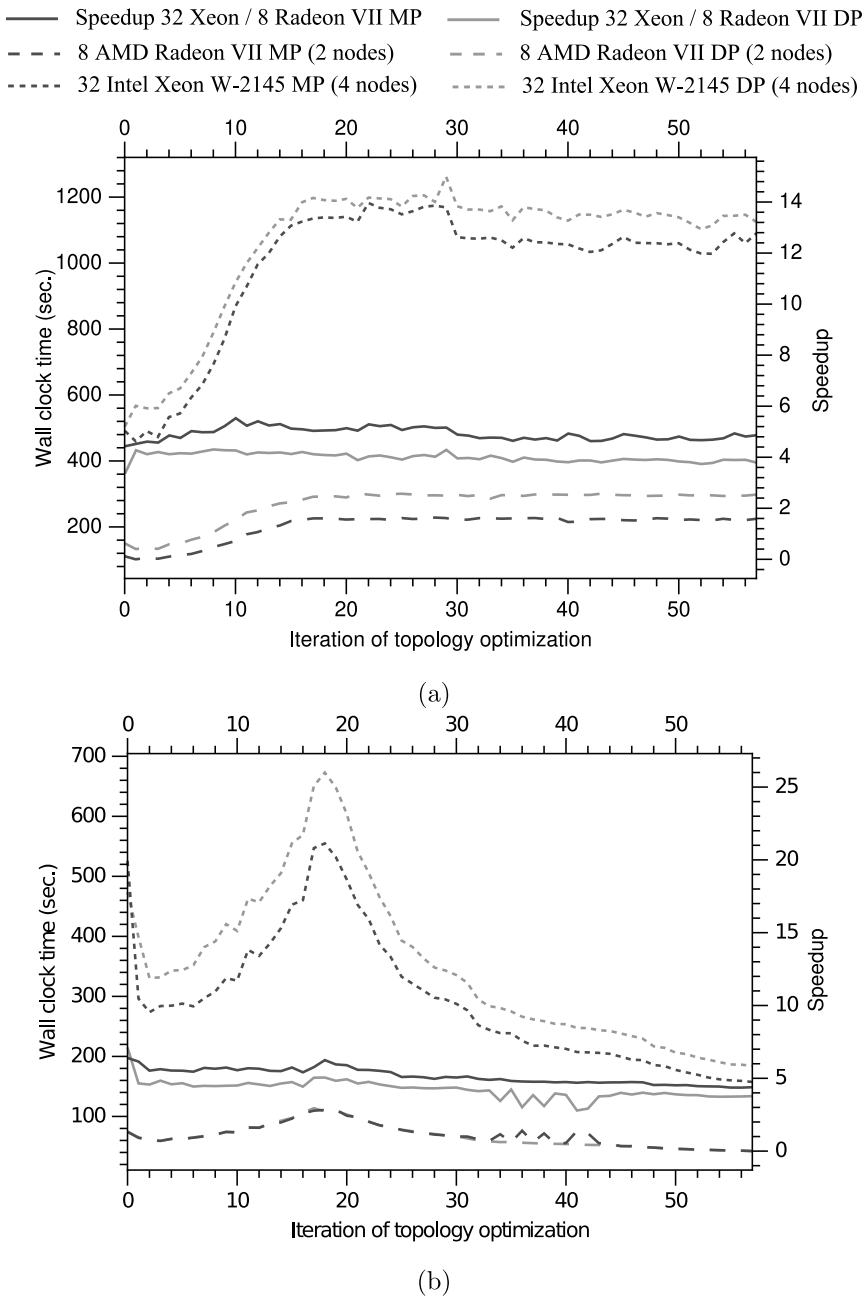


Fig. 11. Wall-clock time for solving FEA of the wheel rim experiment using 88 million tetrahedra finite elements (45023040 unknowns): with tolerance (a) 10^{-8} and (b) 10^{-6} reusing the solution of the previous iteration.

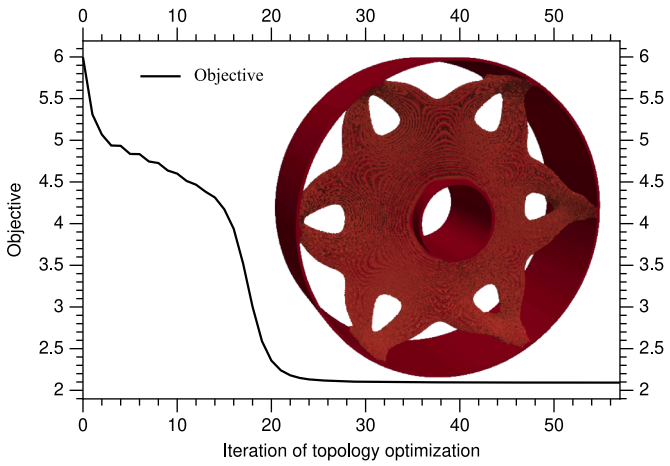


Fig. 12. Evolution of the objective function during the topology optimization of the wheel rim experiment with 88 million tetrahedra finite elements (45023040 unknowns).

solving FEA using a tolerance $tol = 10^{-6}$ and reusing the result of the displacement field of the last optimization iteration as the initial seed of the iterative solver. We run all the experiments using multi- and many-core computing with mixed-precision. We obtain a speedup of around 6.1x using many-core computing with 8 AMD Radeon VII (2 compute nodes) in comparison with multi-core computing with 32 cores of Intel Xeon W-2145 (4 compute nodes). In this experiment, we observe a moderate reduction of the speedup reducing the tolerance of the iterative solver. Figure 18 shows the final design and the evolution of the objective function during the topology optimization, which is similar using the tolerances mentioned above. It also depicts a suitable intermediate result running 80 iterations of topology optimization. Thus, we can reduce a high number of topology optimization iterations by adjusting the stopping criteria of topology optimization.

5.5. Discussion

We have presented an extensive battery of numerical experiments using structured and unstructured meshes with different finite elements. We focus on the FEA solving stage to analyze the performance, feasibility, and scalability of the proposal. Figure 19 justifies that the solving of FEA has been in the spotlight of the study. This figure shows the percentage of wall-clock time spent by the FEA solving stage, the solving of the Helmholtz PDE acting as a sensitivity filter, and the other stages of the topology optimization process using multi-core computing with 32 cores and mixed-precision computing. We take multi-core implementation as a reference because solving FEA shows the worst speedup using many-core computing in the stages mentioned above. We observe that the solving of Helmholtz PDE takes around 8% of the wall-clock time of the cantilever experiment with 24961923 unknowns, whereas this percentage is reduced significantly for problems with a higher number of unknowns. In particular, it takes about 6,5% of the total wall-clock time in the wheel rim experiment with 45023040 unknowns and is negligible for the hollow circular shaft experiments.

We can draw similar conclusions for the other stages of the topology optimization process. In particular, the distributed computing of the objective function (3), the sensitivities (5), and the update of elemental densities (9). The computation of these stages is embarrassingly parallel, and we can obtain significant speedups using many-core computing. Nevertheless, the bottleneck is the solving of the linear elasticity problem using the finite element method. This problem is exacerbated for large-scale problems, where the wall-clock time spent to perform other stages is negligible.

The models using tetrahedral elements have a higher number of finite elements than the models using hexahedral finite elements. The number of design variables of the optimization algorithm corresponds to the number of finite elements, and thus using meshes of tetrahedra allows us to capture more details in the topology optimization. The numerical experiments using many-core computing have shown significant speedups in comparison with an efficient multi-core implementation. Besides, the use of multiple GPUs allows us to accelerate solving and to increase the problem size that we can address. We observe that the use of structured meshes of tetrahedra accelerates the convergence. We solve the wheel rim experiment with 88 million elements taking 1 hour and 15 minutes using many-core computing with two compute nodes, whereas it requires 6 hours and 40 minutes to solve it using multi-core computing with four compute nodes. We optimize the hollow circular shaft experiment with 106 million elements in one day and one-hour using many-core computing with two compute nodes, whereas we require six days to do it using multi-core computing. Nevertheless, we can solve this experiment in 5 hours using many-core computing by adjusting the stopping criteria of topology optimization and thus reducing the number of topology optimization iterations.

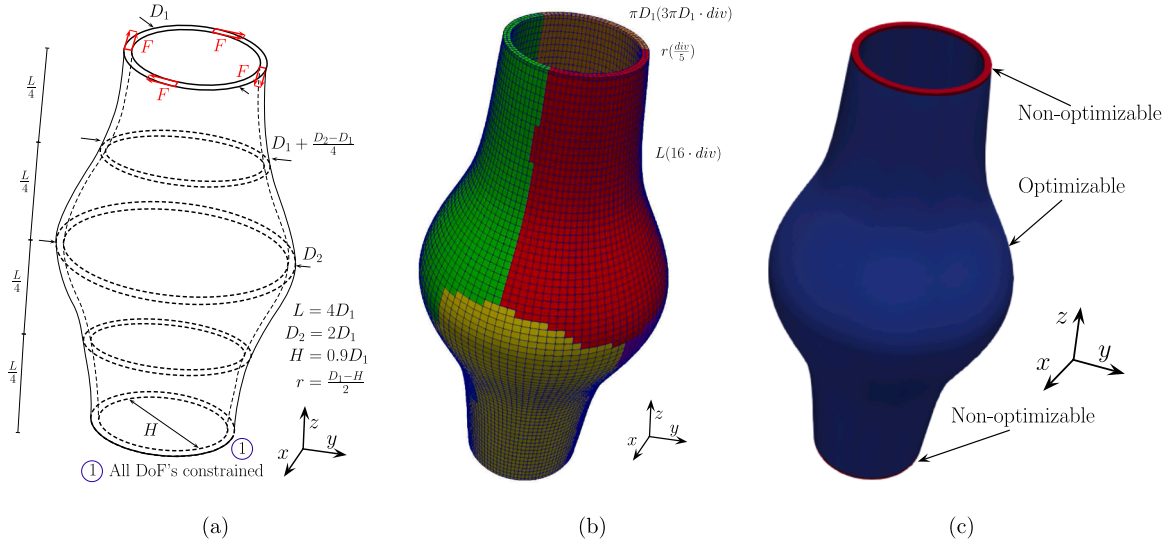


Fig. 13. Hollow circular shaft with varying cross-section experiment: (a) geometric configuration and boundary conditions, (b) mesh parameterization and partitioning into six subdomains, and (c) definition of optimizable and non-optimizable volumes.

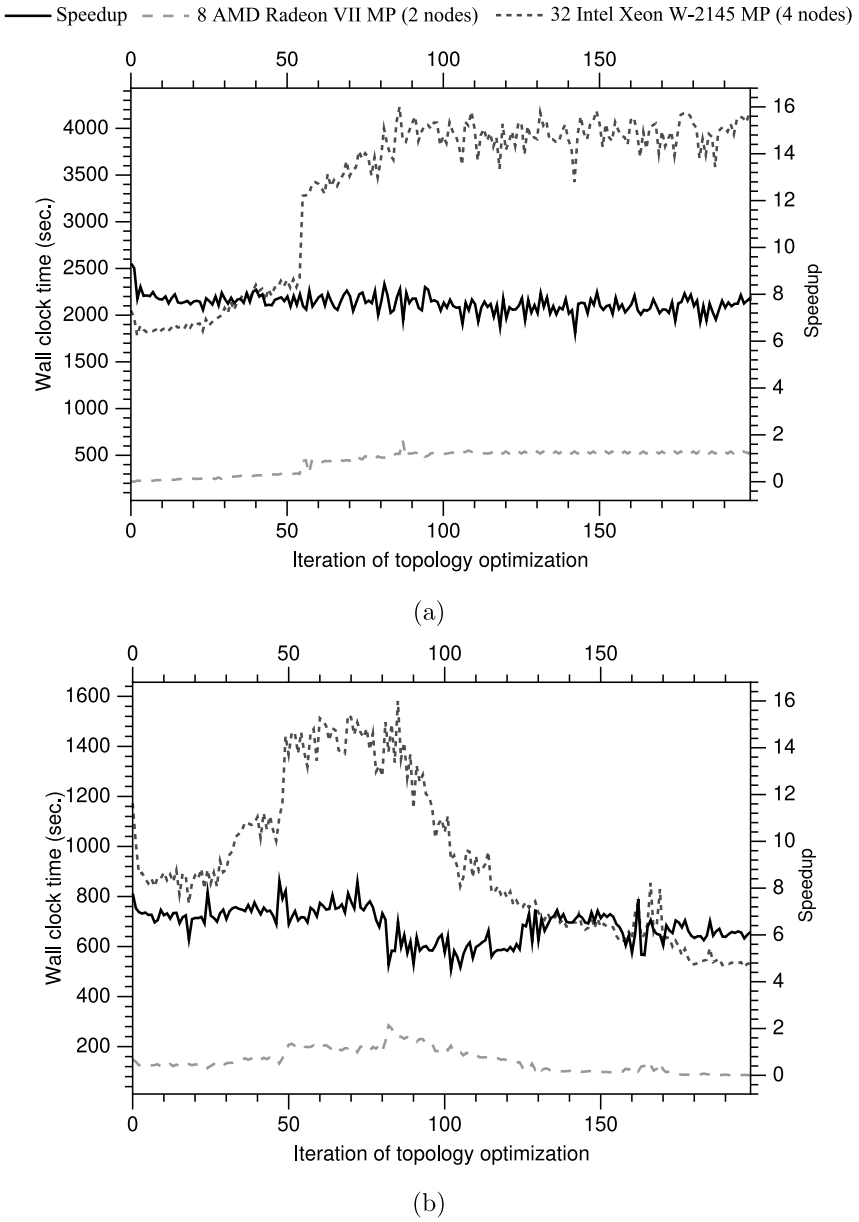


Fig. 14. Wall-clock time for solving FEA of the hollow circular shaft with varying cross-section experiment using 15 million hexahedra finite elements (49860864 unknowns): with tolerance (a) 10^{-8} and (b) 10^{-6} reusing the solution of the previous iteration.

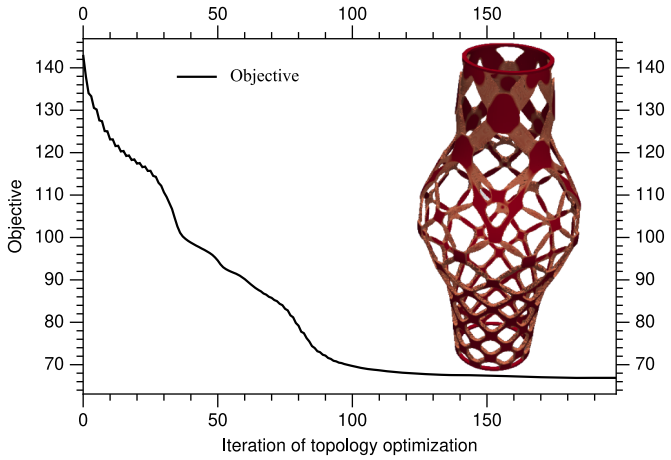


Fig. 15. Evolution of the objective function during the topology optimization of the hollow circular shaft with varying cross-section experiment with 15 million hexahedra finite elements (49860864 unknowns).

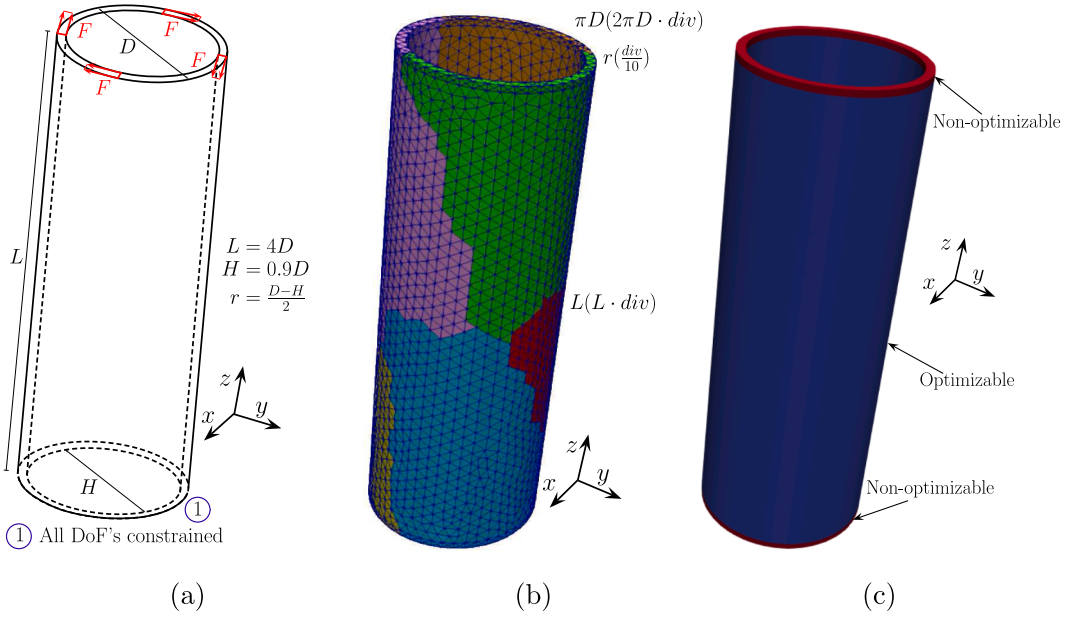


Fig. 16. Hollow circular shaft experiment: (a) geometric configuration and boundary conditions, (b) mesh parameterization and partitioning into six subdomains, and (c) definition of optimizable and non-optimizable volumes.

The experiments using hexahedral elements can address problems with a fewer number of finite elements. In particular, we solve the cantilever experiment with 8 million finite elements in 3 hours and 10 minutes using many-core computing with two compute nodes, requiring 10 hours using multi-core computing with four compute nodes. Finally, we solve the hollow circular shaft with a varying cross-section experiment with 15 million finite elements in 9 hours using many-core computing with two compute nodes, requiring two-days and eight-hours using multi-core computing with four compute nodes.

6. Conclusion

We have presented a parallel implementation of density-based topology optimization using a distributed GPU system. We use non-overlapped partitioning of the domain into balanced subdomains to distribute the computation of density-based topology optimization stages to the computation resources. The solving of FEA in the iterations of topology optimization is the well-known bottleneck of such a problem. The numerical results show that using a distributed conjugate gradient solver preconditioned with aggregate AMG is rewarding using GPU computing. This fact is particularly the case for large-scale problems because smooth aggregation-based AMG methods define interpolation as a piecewise constant operator with only one non-zero per row. This fact makes that we need fewer levels in the V-cycles than classical AMG, which reduces memory consumption drastically and is rewarding for GPU computing. The use of multiple GPU devices allows to accelerate the computing process and to increase the available device memory. This fact enables us to address large-scale topology optimization problems that commonly do not fit on one GPU. The adopted techniques aim to accelerate the solving stage dominating the overall speedup of the topology optimization problem.

We test the influence of the tolerance and the initialization of the distributed conjugate gradient showing that the performance is highly dependent on such parameters obtaining similar topology optimization designs. This fact makes difficult the performance comparison with other approaches. We evaluate the strong and weak scaling of the GPU

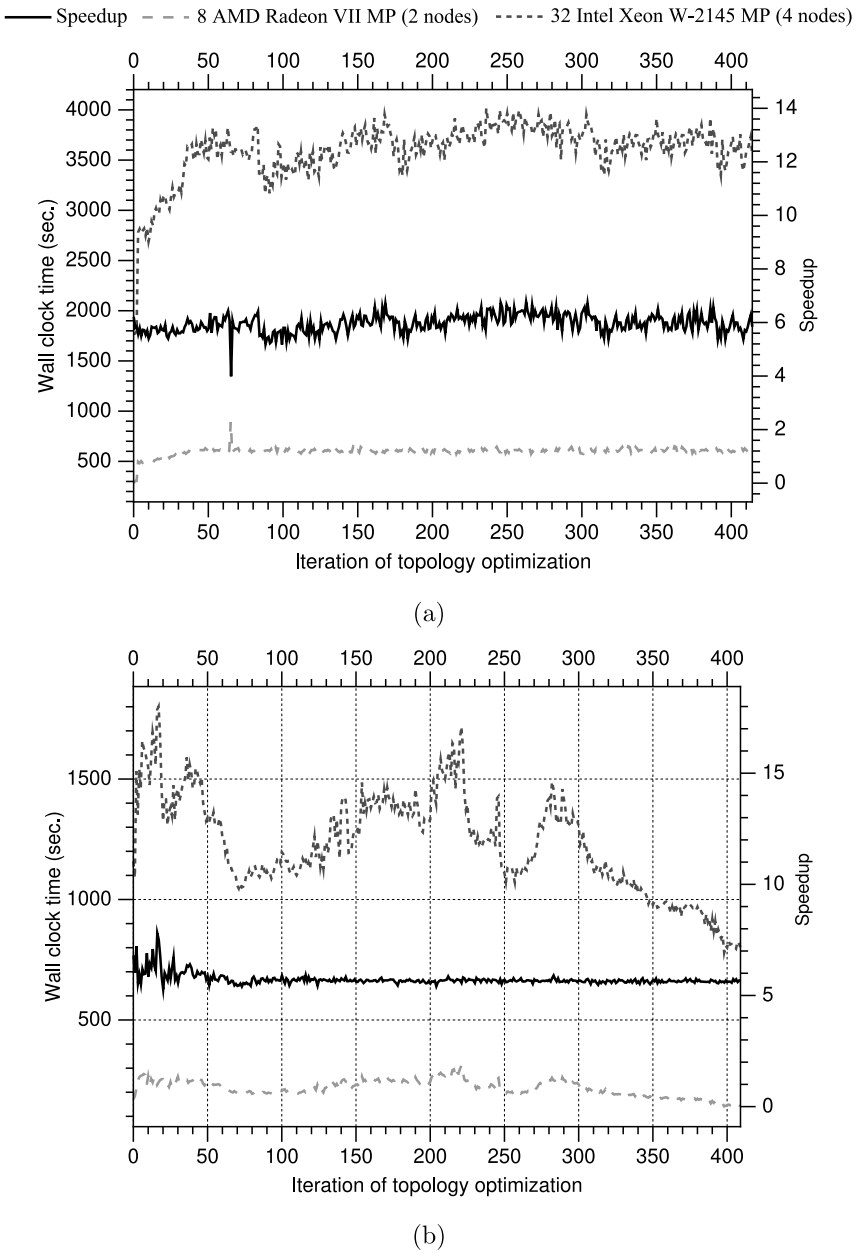


Fig. 17. Wall-clock time for solving FEA of the hollow circular shaft experiment using 106 million tetrahedra finite elements (56513256 unknowns): with tolerance (a) 10^{-8} and (b) 10^{-6} reusing the solution of the previous iteration.

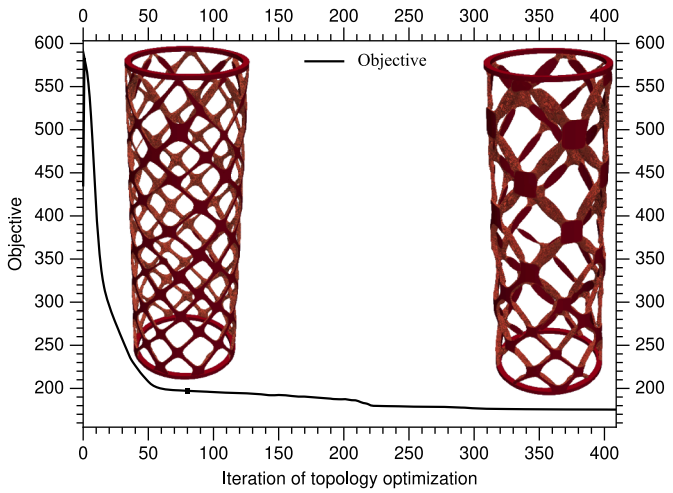


Fig. 18. Evolution of the objective function during the topology optimization of the hollow circular shaft experiment with 106 million tetrahedra finite elements (56513256 unknowns).

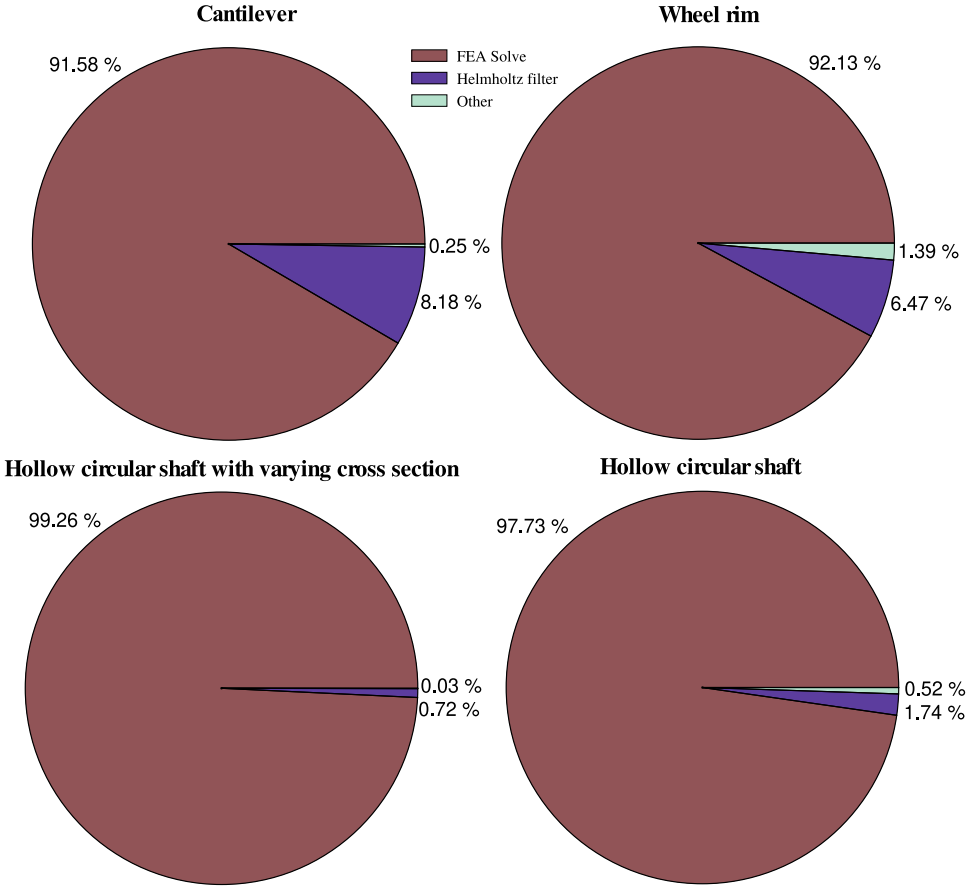


Fig. 19. Percentage of wall-clock time for solving the different stages of topology optimization.

implementation showing good scalability as increasing the number of compute nodes and poor performance as reducing the problem size for each subdomain. This fact is a well-known issue in GPU computing, where the problem size should be high enough to make significant computing in comparison with data transfer between host and device memory. We evaluate the performance improvement using mixed-precision for the AMG setup and V-cycles of the preconditioner. We also test the proposal using structured and unstructured meshes with different finite elements and relaxing operators, achieving significant speedups using many-core computing with low-cost GPUs (two compute nodes) in comparison with an efficient multi-core implementation (four compute nodes).

Declaration of Competing Interest

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

Acknowledgment

This work has been supported by the AEI/FEDER and UE under the contract DPI2016-77538-R, and by the "Fundación Séneca – Agencia de Ciencia y Tecnología de la Región de Murcia" of Spain under the contract 20911/PI/18.

References

[1] Deaton JD, Grandhi RV. A survey of structural and multidisciplinary continuum topology optimization: post 2000. Struct Multidiscip Optim 2014;49(1):1–38.
[2] Allaire G, Jouve F, Toader A-M. Structural optimization using shape sensitivity analysis and a level-set method. J Comput Phys 2004;194:363–93.
[3] Takezawa A, Nishiwaki S, Kitamura M. Shape and topology optimization based on the phase field method and sensitivity analysis. J Comput Phys 2010;229:2697–718.
[4] Christiansen AN, Nobel-Jørgensen M, Aage N, Sigmund O, Bærentzen JA. Topology optimization using an explicit interface representation. Struct Multidisc Optim 2014;49(3):387–99.
[5] Bendsøe MP, Kikuchi N. Generating optimal topologies in structural design using a homogenization method. Comput Meth Appl Mech Eng 1988;71(2):197–224.
[6] Zhou M, Rozvany GIN. The COC algorithm, Part II: Topological, geometrical and generalized shape optimization. Comput Methods Appl Mech Eng 1991;89(1):309–36.
[7] Bendsøe MP, Sigmund O. Topology Optimization – Theory, Methods, and Applications. second. Springer-Verlag Berlin Heidelberg; 2004.
[8] Venkataraman S, Haftka R. Structural optimization complexity: what has Moore's law done for us? Struct Multidiscip Optim 2004;28(6):375–87.
[9] Nguyen T-H, Paulino G-H, Song J, Le C-H. A computational paradigm for multiresolution topology optimization (MTOP). Struct Multidiscip Optim 2010;41(4):525–39.
[10] Liu H, Wang Y, Zong H, Wang M-Y. Efficient structure topology optimization by using the multiscale finite element method. Struct Multidiscip Optim 2018;58:1411–30.
[11] Gupta D-K, van Keulen F, Langelaar M. Design and analysis adaptivity in multiresolution topology optimization. Int J Numer Methods Eng 2020;121(3):450–76.
[12] Wang S, de Sturler E, Paulino G-H. Large-scale topology optimization using preconditioned Krylov subspace methods with recycling. Int J Numer Methods Eng 2007;69(12):2441–68.
[13] Amir O, Stolpe M, Sigmund O. Efficient use of iterative solvers in nested topology optimization. Struct Multidiscip Optim 2010;42(1):55–72.
[14] Amir O, Aage N, Lazarov B-S. On multigrid-CG for efficient topology optimization. Struct Multidiscip Optim 2014;49(1):815–29.
[15] Amir O, Bendsøe MP, Sigmund O. Approximate reanalysis in topology optimization. Int J Numer Methods Eng 2009;78(12):1474–91.
[16] Mo K, Guo D, Wang H. Iterative reanalysis approximationassisted moving morphable componentbased topology optimization method. Int J Numer Methods Eng 2020;121(22):5101–22.
[17] Liu J, Li E, Wu Y, Wang H. An efficient auxiliary projection-based multigrid isogeometric reanalysis method and its applicationin an optimization framework. Int J Numer Methods Eng 2020;121(13):2857–73.
[18] Borrvall T, Petersson J. Large-scale topology optimization in 3D using parallel computing. Comput Methods Appl Mech Eng 2001;190(46–47):6201–29.
[19] Vemaganti K, Lawrence W-E. Parallel methods for optimality criteria-based topology optimization. Comput Methods Appl Mech Eng 2005;194(34–35):3637–67.
[20] Aage N, Andreassen E, Lazarov B-S. Topology optimization using PETSc: An easy-to-use, fully parallel, open source topology optimization framework. Struct Multidiscip Optim 2015;51(3):565–72.
[21] Aage N, Andreassen E, Lazarov B-S, Sigmund O. Giga-voxel computational morphogenesis for structural design. Nature 2017;550(84):84–6.
[22] Liu H, Tian Y, Zong H, Ma Q, Wang M-Y, Zhang L. Fully parallel level set method for large-scale structural topology optimization. Comput Struct 2019;221:13–27.
[23] Kahle J, Moreno J, Dreps D. Summit and Sierra: Designing AI/HPC Supercomputers. Proc. of IEEE Int. Solid-State Circuits Conference. 2019. p. 42–3. San Francisco, CA, USA
[24] Nickolls J, Dally W. The GPU Computing Era. IEEE Micro 2010;30(2):56–69.
[25] Noaje G, Krajecki M, Jaillet C. MultiGPU computing using MPI or OpenMP. Int. Conf. on Intelligent Computer Communication and Processing. 2010. p. 347–54. Cluj-Napoca, Romania
[26] Wadbro E, Berggren M. Megapixel Topology Optimization on a Graphics Processing Unit. SIAM Rev 2009;51(4):707–21.
[27] Schmidt S, Schulz V. A 2589 line topology optimization code written for the graphics card. Comput Vis Sci 2011;14(6):249–56.
[28] Martínez-Frutos J, Martínez-Castejón P, Herrero-Pérez D. Efficient topology optimization using GPU computing with multilevel granularity. Adv Eng Softw 2017;106:47–62.
[29] Martínez-Frutos J, Martínez-Castejón PJ, Herrero-Peréz D. Fine-grained GPU implementation of assembly-free iterative solver for finite element problems. Comput Struct 2015;157:9–18.
[30] Dick C, Georgii J, Westermann R. A Real-Time Multigrid Finite Hexahedra Method for Elasticity Simulation using CUDA. Simulation Modelling Practice and Theory 2011;19(2):801–16.
[31] Suresh K. Efficient generation of large-scale pareto-optimal topologies. Struct Multidiscip Optim 2013;47(1):49–61.
[32] Wu J, Dick C, Westermann R. A System for High-Resolution Topology Optimization. IEEE Trans Visual Comput Graphics 2016;22(3):1195–208.
[33] Martínez-Frutos J, Herrero-Pérez D. GPU acceleration for evolutionary topology optimization of continuum structures using isosurfaces. Comput Struct 2017;182:119–36.
[34] Gavranovic S, Hartmann D, Wever U. Topology Optimization Using GPGPU. Springer; 2019. p. 553–66.
[35] Dambrine M, Kateb D. On the ersatz material approximation in level-set methods. ESAIM: Control, Optimisation and Calculus of Variations 2010;16(3):618–34.
[36] Ramírez-Gil F-J, Nelli-Silva E, Montealegre-Rubio W. Topology optimization design of 3D electrothermomechanical actuators by using GPU as a co-processor. Comput Methods Appl Mech Eng 2016;302:44–69.
[37] Zegard T, Paulino G. Toward GPU accelerated topology optimization on unstructured meshes. Struct Multidiscip Optim 2013;48(3):473–85.
[38] He G, Wang H, Li E, Huang G, Li G. A multiple-GPU based parallel independent coefficient reanalysis method and applications for vehicle design. Adv Eng Softw 2015;85:108–24.
[39] Jung J-H, Bae D-S. An improved direct linear equation solver using multi-GPU in multi-body dynamics. Adv Eng Softw 2018;115:87–102.
[40] Martínez-Frutos J, Herrero-Peréz D. Large-scale robust topology optimization using multi-GPU systems. Comput Methods Appl Mech Eng 2016;311:393–414.
[41] Papadrakakis M, Stavroulakis G, Karatarakis A. A new era in scientific computing: Domain decomposition methods in hybrid CPU–GPU architectures. Comput Meth Appl Mech Eng 2011;200(13-16):1490–508.
[42] Bendsøe MP, Sigmund O. Material interpolation schemes in topology optimization. Arch Appl Mech 1999;69(9–10):635–54.
[43] Lazarov B-S, Sigmund O. Filters in topology optimization based on helmholtz-typedifferential equations. Int J Numer Methods Eng 2011;86(6):765–81.
[44] Wallin M, Ivarsson N, Amir O, Tortorelli D. Consistent boundary conditions for pde filter regularization in topology optimization. Struct Multidiscip Optim 2020;62:1299–311.
[45] Bendsøe MP. Optimization of structural topology shape and material. Springer, New York; 1995.
[46] Bitzarakis S, Papadrakakis M, Kotsopulos A. Parallel solution techniques in computational structural mechanics. Comput Methods Appl Mech Eng 1997;148(1–2):75–104.
[47] Karypis G, Kumar V. Multilevel k-way partitioning scheme for irregular graphs. J Parallel Dist Com 1998;48(1):96–129.
[48] Karypis G, Schloegel K. ParMeTis: Parallel Graph Partitioning and Sparse Matrix Ordering Library, Version 4.0. Tech. Rep. University of Minnesota, Minneapolis, MN; 2013.
[49] Gandham R, Esler K, Zhang Y. A GPU accelerated aggregation algebraic multigrid method. Comput Math Appl 2014;68(10):1151–60.
[50] Demidov D. AMGCL: An Efficient, Flexible, and Extensible Algebraic Multigrid Implementation. Lobachevskii J Math 2019;40(5):535–46.
[51] Demidov D, Ahnert K, Rupp K, Gottschling P. Programming CUDA and OpenCL: A Case Study Using Modern C++ Libraries. SIAM J Sci Comput 2013;35(5):C453–72.
[52] Karsten A, Demidov D, Mulansky M. Solving Ordinary Differential Equations on GPUs. Cham: Springer International Publishing; 2014. p. 125–57.
[53] Abhyankar S., Brown J., Constantinescu E.M., Ghosh D., Smith B.F., Zhang H.. PETSc/TS: A Modern Scalable ODE/DAE Solver Library. arXiv:180601437 2018;.
[54] Ruge J, Stüben K. Algebraic Multigrid. In: Ewing R, editor. Multigrid Methods. 3600 University City Science Center, Philadelphia, Pennsylvania: Society for Industrial and Applied Mathematics (SIAM); 1987. p. 73–130.