Composite Structures 107 (2014) 288–297

Contents lists available at ScienceDirect

Composite Structures

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

# A hybrid shared/distributed memory parallel genetic algorithm for optimization of laminate composites

I.B.C.M. Rocha, E. Parente Jr. [*], A.M.C. Melo

*Laboratório de Mecânica Computacional e Visualização (LMCV), Universidade Federal do Ceará, Campus do Pici, Bloco 728, 60455-760 Fortaleza, Ceará, Brazil*

ARTICLE INFO

*Article history:*
Available online 12 August 2013

*Keywords:*
Composite materials
Genetic algorithms
Parallel computing
FEM

ABSTRACT

This work presents a genetic algorithm combining two types of computational parallelization methods, resulting in a hybrid shared/distributed memory algorithm based on the island model using both OpenMP and MPI libraries. In order to take further advantage of the island configuration, different genetic parameters are used in each one, allowing the consideration of multiple evolution environments concurrently. To specifically treat composite structures, a three-chromosome variable encoding and special laminate operators are used. The resulting gains in execution time due to the parallel implementation allow the use of high fidelity analysis procedures based on the Finite Element Method in the optimization of composite laminate plates and shells. Two numerical examples are presented in order to assess the performance and reliability of the proposed algorithm.

## 1. Introduction

The use of laminated composite structures has been growing in the last few years. In such materials, composite laminas (plies) are stacked, offering high tailorability, since the number, orientation, thickness and sequence of plies can be optimized to give the exact desired structural behavior. Each lamina consists of unidirectional fibers embedded in a polymeric matrix. However, due to the large number of design variables, including number of layers, thickness, fiber orientation and material of each layer, the traditional trial-and-error procedure can be arduous in the design of laminated structures, thus requiring the use of optimization techniques [1].

In the optimization of laminated structures, the design variables can be discrete [2] or continuous [3]. To solve the optimization problem, evolutionary algorithms have seen extensive use, particularly Genetic Algorithms (GAs) [4–6]. Such algorithms readily deal with discrete variables and do not tend to get trapped in local minima.

In order to evaluate the performance of a given design in an optimization procedure, the mechanical response of the structure has to be evaluated through an analysis procedure. The level of fidelity of such analysis, defined as how well the real structure is represented by the idealized mechanical model, is an important factor in the quality of the obtained designs. As many optimization tools, particularly GAs, are computationally expensive, the structural analysis is usually carried out using analytical solutions. Such methods, albeit fast, can only be applied for simple geometries, support conditions and load configurations, thus begin unable to accurately represent real structures. Yet, most of the current work regarding laminate structure optimization rely on such solutions, resulting in limited applicability.

In this regard, numerical methods such as the Finite Element Method (FEM) are used to analyze complex structures. However, even within FEM, multiple levels of fidelity can be chosen, concerning not only the mesh size but also the element type, its cinematic assumptions and whether or not non-linearities are taken into account. Unfortunately, such increases in fidelity also lead to higher computational costs, which may render the analysis procedure infeasible to use with optimization techniques such as GAs. Thus, parallelization techniques [7] are used to make the optimization algorithms faster. Some recent works have been successfully applying such methods in laminate composite optimization [8,9].

Many different parallelization techniques can be applied to GAs. In this work, two of them will be combined, one suitable for shared memory architectures and the other suitable for distributed memory ones. The result is a hybrid parallel GA which can be executed in personal computers and high-performance cluster computers alike. This is achieved by combining the simplicity of OpenMP [10] for shared memory architectures with the flexibility of MPI [11] for distributed memory architectures. The resultant algorithm uses an island model, working with many subpopulations evolving concurrently. Also, seeking to further improve convergence, each island evolves with a distinct set of genetic parameters, effectively introducing different evolution environments.

In Section 2, the parallel genetic algorithm is formulated and its implementation is briefly discussed. In Section 3, two numerical

* Corresponding author. Tel.: +55 85 3366 9198.
*E-mail addresses:* iuribarcelos@yahoo.com.br (I.B.C.M. Rocha), evandro@ufc.br (E. Parente Jr.), macario@ufc.br (A.M.C. Melo).

0263-8223/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.compstruct.2013.07.049

examples are presented to assess the performance and reliability of the algorithm and the conclusions are given in Section 4.

## 2. Genetic algorithm

In nature, individuals must have useful genetic features in order to survive. Such individuals are the fittest and thus have more chances of passing their characteristics to future generations, while those less fit succumb along the evolution process.

GAs present many advantages that have led to a rise in their use, particularly with the rise in the processing power of computers. As they work with a set of potential solutions, GAs do not easily get trapped in local minima. Furthermore, since no gradient information is needed, GAs can deal with discontinuous design spaces and problems with discrete variables and non-differentiable functions.

One of the main drawbacks of working with GAs is their high computational cost. Particularly when the optimization problem involves complex implicitly defined objective function and constraints or a high number of design variables, it may take hours to run a single optimization process.

Thus, techniques of parallel computing are very appealing in order to reduce the total computational cost of GAs. In this regard, it is important to develop parallel implementations not only for high-cost cluster computers but also for low-cost personal computers with multi-core architectures. The proposed algorithm is presented in the following.

### 2.1. Encoding

Each trial laminate is an individual represented by three chromosomes, each being a line in a matrix whose columns represent the laminate layers. The *thickness chromosome* stores information about thickness, the *orientation chromosome* stores data about the fiber orientation angles and the *material chromosome* stores material properties, allowing the design of hybrid laminates.

Each laminate has a genotypical and phenotypical representation. On the genotype, chromosomes have integer values in each gene, corresponding to list positions which store discrete values of the design variables. Thus, the algorithm used in this work features integer coding. The phenotype representation is used to store the actual values of the design variables.

The number of columns in the genotype matrix depends on the laminate type. If a general laminate is considered, the number of columns is fixed at the maximum number of layers. If, however, a symmetric laminate is used, the number of columns will be half the maximum number of layers. Finally, if a symmetric and balanced laminate is used, the number of columns will be only one quarter of the maximum number of layers.

Genetic operators are applied to the individuals in their genotype representation, which in turn has to be decoded in order to evaluate the objective function and constraints. This process is shown in Fig. 1.

When layers are deleted, their thickness value will be set to zero and they will be ignored in the decoding process. Consequently, the number of columns in the phenotype representation will depend on the number of deleted layers.

### 2.2. Fitness function and selection

For the purpose of selection to the mating pool and subsequent selection for crossover, individuals need be assigned a positive numerical value that represents its fitness in the current population. Such value dictates how an individual is better or worse than others.

Here, as only minimization problems are treated, the fitness function is defined by mapping the penalized objective function in order to ensure that the selection probabilities are positive and that the best individuals have the highest fitness values [4]. The penalized objective function is evaluated using the adaptive penalty method developed by Barbosa and Lemonge [12], using no user-defined parameters. The values are then mapped by:

$$Fit_i = max(|f_{p(min)}|, |f_{p(max)}|) - f_{pi} \tag{1}$$

where $f_{p(min)}$ and $f_{p(max)}$ are the minimum and maximum penalized objective function values in the current population, respectively.

In the present work, the classic Roulette selection method is used [1] with a Fitness Proportional Selection (FPS) strategy. Thus, the probability of selection ($p$) is directly proportional to the fitness value:

$$p = \frac{Fit_i}{\sum_i^{N_{ind}} Fit_i} \tag{2}$$

### 2.3. Crossover

Crossover is a key genetic operator for GA convergence. It is applied on two individuals, called parents, and originates two new individuals called sons, which contain the combined traits of the parents. Parents are taken from the mating pool, which is filled with individuals of the original population, using the selection process introduced in Section 2.2. The number of parents selected for crossover is dictated by the *crossover rate* ($r_c$).

Generally, the crossover operator in laminate structures optimization with integer variables consists in the definition of one or two crossover points and the recombination of the resultant portions of the laminate of each parent to form the sons [2,13,14]. In this work, a new method based on the crossover used in real-coded GAs is proposed for the case of integer variables. It seems to the authors that this method provides higher individual variability and less tendency for premature convergence.

The method consists of a linear combination of the parents genes in order to form the sons. The process is done layer by layer, with a random $r$ number between 0 and 1 been chosen. Such random number is then used to calculate a weighted mean of the parents genes, rounded to the closest integer. Fig. 2 shows an example of the process.

### 2.4. Mutation

Even though genetic algorithms have less chance of getting trapped in local minima, sometimes a premature convergence can occur. To prevent it, genetic variability has to be maintained. The mutation operator is one of the strategies used to ensure variability within the population and design space exploration. Mutation is applied in the offspring generated by the crossover with a *mutation probability* $p_m$ to which low values are usually assigned. For each layer of the chromosome, a random number between 0 and 1 ($r$) is generated. If such number is lower than the *mutation probability*, a feasible random integer replaces the original gene value. Fig. 3 shows mutation applied on two genes.

### 2.5. Laminate operators

The gene-swap, or layer-swap operator changes the mechanical properties of the laminate by exchanging the position of two layers (Fig. 4). As with the mutation operator, the layer-swap generally has a low probability of occurrence, called *swap probability* ($p_s$). The choice of plies to be exchanged is also randomized. It is important to note that the layer-swap operator changes the bending

| Code | Thickness Values |
|---|---|
| 0 | 0.0mm |
| 1 | 0.1mm |
| 2 | 0.2mm |
| 3 | 0.3mm |
| 4 | 0.4mm |

| Code | Angle Values |
|---|---|
| 1 | -45° |
| 2 | -25° |
| 3 | -5° |
| 4 | +25° |
| 5 | +45° |

| Code | Materials |
|---|---|
| 1 | Material 1 |
| 2 | Material 2 |
| 3 | Material 3 |
| 4 | Material 4 |
| 5 | Material 5 |

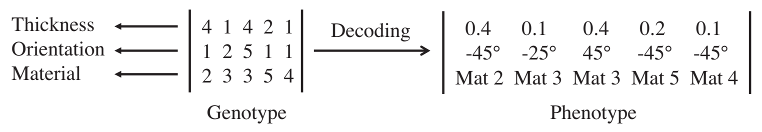


**Fig. 1.** Genotype to phenotype decoding example.

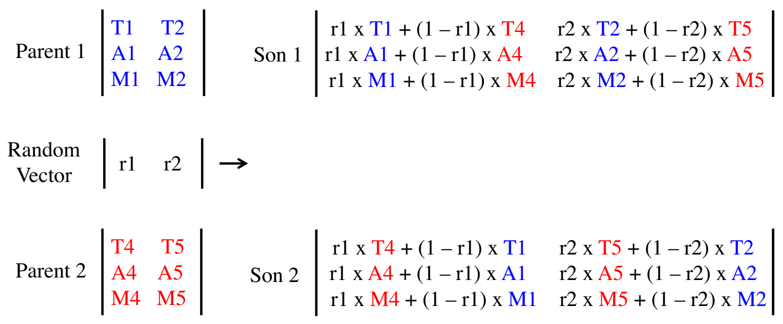

Parent 1
T1 T2
A1 A2
M1 M2
Son 1
r1 x T1 + (1 – r1) x T4 r2 x T2 + (1 – r2) x T5
r1 x A1 + (1 – r1) x A4 r2 x A2 + (1 – r2) x A5
r1 x M1 + (1 – r1) x M4 r2 x M2 + (1 – r2) x M5
Random Vector
r1 r2
Parent 2
T4 T5
A4 A5
M4 M5
Son 2
r1 x T4 + (1 – r1) x T1 r2 x T5 + (1 – r2) x T2
r1 x A4 + (1 – r1) x A1 r2 x A5 + (1 – r2) x A2
r1 x M4 + (1 – r1) x M1 r2 x M5 + (1 – r2) x M2

**Fig. 2.** Crossover operator applied on two 2-layer laminates.

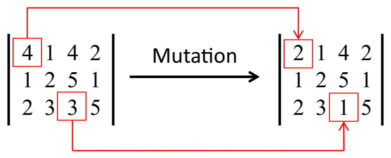


**Fig. 3.** Mutation operator applied on two genes of a 4-layer chromosome.

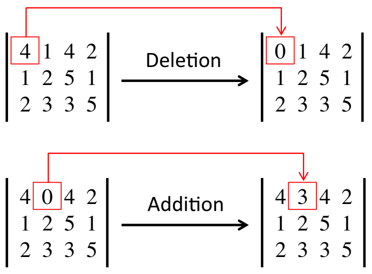


**Fig. 5.** Layer addition and deletion.

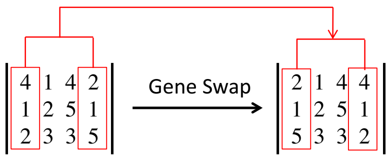


**Fig. 4.** Example of the layer-swap operator.

stiffness of the laminate without changing its membrane stiffness, which can be useful in structural applications involving bending and buckling [15].

The ply deletion operator is similar to a mutation. For each layer, a random number between 0 and 1 is generated. If it is smaller than the *deletion probability* ($p_d$), the thickness of such layer is set as 0, thus effectively eliminating the layer. Similarly, the ply addition is applied in all genes which were previously deleted. If the random number is smaller than the *addition probability* ($p_a$), a random thickness and orientation angle are inserted, effectively adding a layer. Fig. 5 shows examples of ply deletion and addition.

## 2.6. Hybrid parallelization

As previously noted, GAs tend to get increasingly time consuming as the individual evaluation procedure is refined. When using FEA or other computationally expensive analysis method, the individual evaluation becomes the most time-consuming part of the algorithm. However, the evaluation of a given individual depends only on its own variables and can be readily implemented in parallel [7,16].

Many different parallelization techniques can be used in GAs [7]. In this work, two distinct techniques, Global Parallelization and Coarse-Grain, will be combined to form a hybrid algorithm.

The Global Parallelization scheme is the most simple technique. It is also the one that most resembles a sequential GA. It maintains the concept of a single population in which every individual can mate and compete with all others [7] in a *panmictic* approach [16]. In its simplest form, only the individual evaluations are executed in parallel. In this work, in addition to the evaluations, other routines such as the genetic operators, constraint handling and fitness function evaluation are also parallelized. Fig. 6 shows the basic idea of the Global Parallelization technique.

This level of shared memory parallelization (Fig. 6) is handled in this work using the OpenMP library [10]. In distributed memory configurations, a master thread has to be explicitly chosen to receive results of tasks distributed to slave threads. When using OpenMP, such tasks are automatically handled, which greatly reduces the implementation effort. This is also the most suitable approach for computers with multi-core processors. Even though this

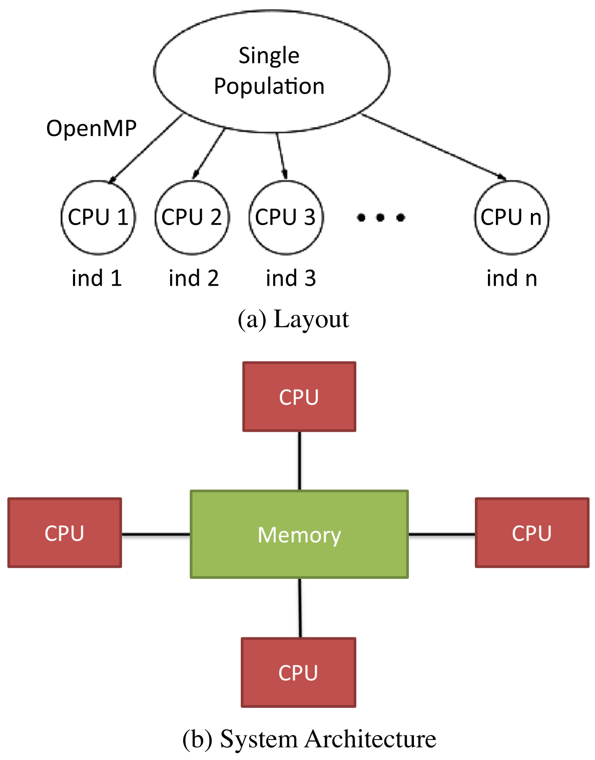


**Fig. 6.** Global Parallelization.

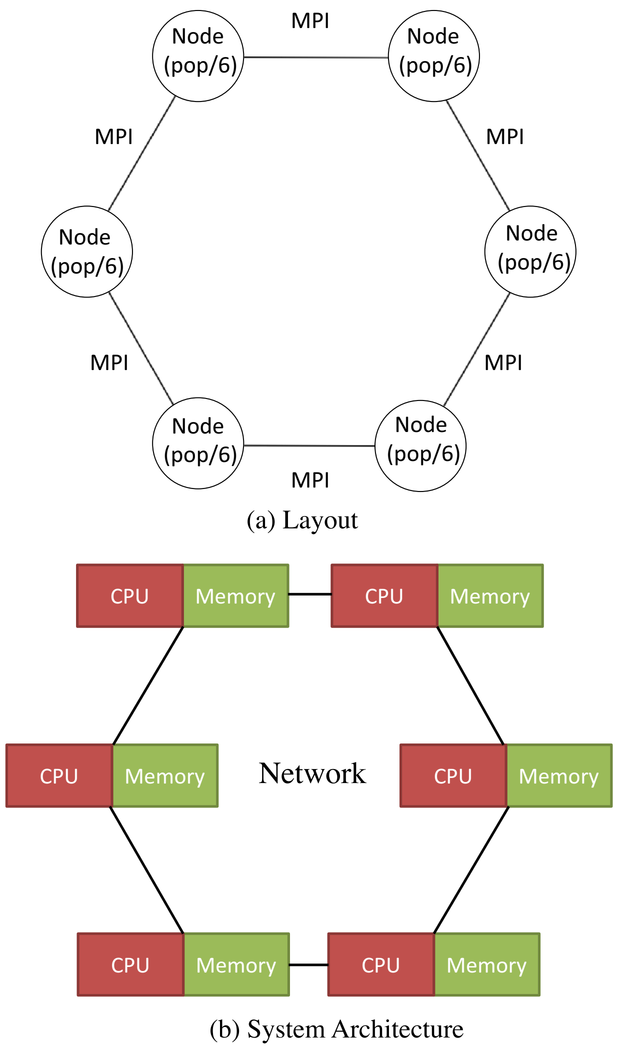


**Fig. 7.** Coarse-grain parallelization.

type of architecture is common nowadays, few developers take advantage of the potential gains of parallelization.

In a coarse-grain approach, also called *Island Model*, the original population is divided in a series of subpopulations called *islands* or *demes* [7]. Such demes evolve in parallel but mostly isolated from each other. Interaction between demes is done by the operator known as *migration*.

In natural evolution, the *Theory of Punctuated Equilibria* states that most of the time a population will not exhibit significant genetic changes and maintain a state of equilibrium or *stasis*, but sudden environment changes can lead to a rapid evolution [7]. This is especially valid in small populations because these beneficial evolutionary changes are not diluted by a large population size. In GAs, a multi-deme configuration with little communication tends to lead to a similar behavior, with the sudden insertion of migrated individuals leading to changes in the evolution environment. This link with natural evolution, while endorsing the use of coarse-grain configurations in GA implementations, was not their original motivation. Such algorithms were first intended to minimize communication costs in parallel implementations [7].

Performance and reliability of coarse-grain GAs are directly affected by three main factors: The *migration interval*, which represents the frequency of migrations, the *topology*, governing the directions to which migrated individuals move, and the *migration rate*, which represents the number of individuals that migrate. Fig. 7 shows a coarse-grain GA arrangement with 6 demes in a ring topology.

Coarse-grain GAs were created with distributed memory configurations in mind and seek a suitable migration process to make its reliability match or surpass that of a panmictic GA with a single large population while minimizing the communication overhead. Here, such distributed memory communications (Fig. 7) were done using the MPI (*Message Passing Interface*) library [11].

The proposed algorithm is a combination of the two above strategies in a hybrid parallelization approach. In the node of a multi-core cluster or in a conventional multi-core personal computer, each subpopulation evolves using a Global Parallelization scheme. When multiple nodes are used, multiple subpopulations are executed in parallel in a coarse-grain scheme with a fully connected topology. Thus, the resulting GA, an example of which is shown in Fig. 8 for a 4-node cluster, mixes the best features of both OpenMP and MPI and can be executed in all modern multi-core architectures.

Taking advantage of the use of multiple subpopulations, ranges are used instead of fixed rates and probabilities for all operators, inspired by what have been done in [17]. That way, each deme has a random rate or probability value for each genetic operator, which is generated inside the specified ranges. This is done in order to avoid the need of tune such parameters for each optimization problem and to consider different evolution environments concurrently, which can lead to better performance.

### 2.7. Load balancing

In the use of the described parallel genetic algorithm, some care has to be taken in order to avoid load balancing problems, i.e. in order to make all processing cores work at their maximum capacity during the whole optimization process. Here, the load is

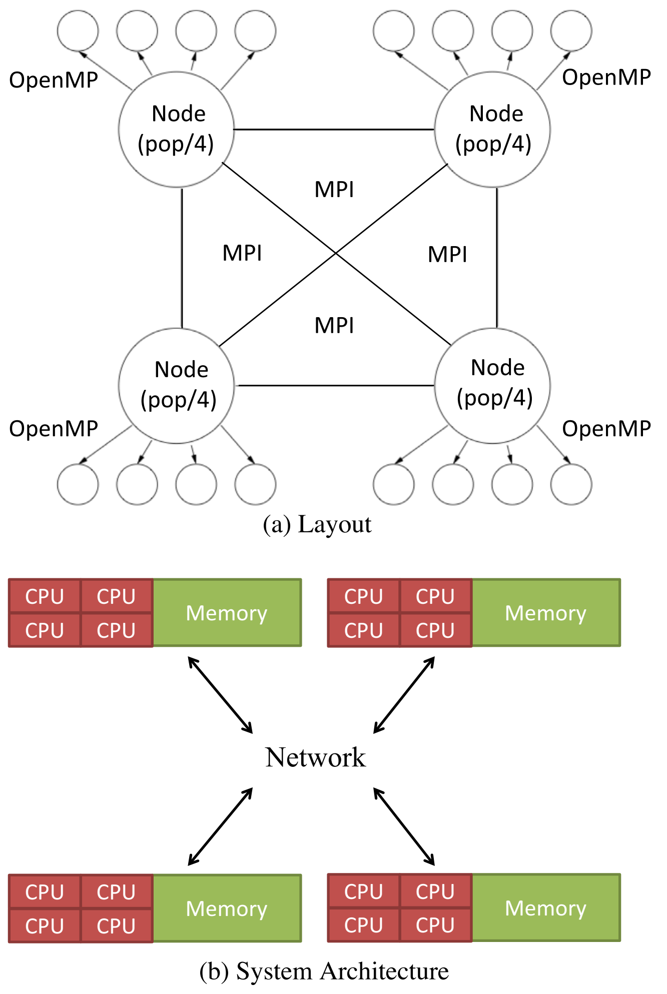


**Fig. 8.** Hybrid parallelization.

statically balanced. The first aspect concerns the population size. For a personal computer with $n$ computing cores or a cluster with $n$ cores per node, the population of each deme has to be a multiple of $n$. If, for instance, a computer has 8 cores and a subpopulation of 9 individuals is used, 7 cores will be idle while the last individual is being evaluated, effectively doubling the optimization time.

Another load balancing aspect concerns the crossover rate. As the number of individuals generated by the crossover operation directly influences the number of individual evaluations, using different crossover rates across multiple demes can severely impact the optimization time. Since the MPI parallelization used here is synchronous, all demes have to be at the same generation for the migration process to begin. Thus, it is important to maintain the crossover rate constant across demes.

It is important to note, however, that changing rates for other genetic operators, such as mutation or gene-swap, does not change the number of individual evaluations. Thus, they are assumed not to have a significant impact on load balancing.

### 2.8. Computational implementation

The proposed algorithm was implemented using the C++ programming language and using the Object Oriented Programming (OOP) paradigm. The class structure includes an *Algorithm* class, which controls the optimization process, creating and manipulating populations, a *Population* class, which creates and manages individuals and an *Individual* class, which contains the relevant genetic operators. Lastly, the *Problem* class handles the objective function and constraint evaluations.

The OpenMP directives were used in loops through individuals inside each deme, such as the objective function evaluation and the genetic operators. Because of the simplicity of OpenMP, a *#pragma* directive is added before the loops and the work is automatically assigned and balanced between processors. However, care was taken to avoid making two individuals write on the same memory simultaneously.

For the MPI parallelization, the operators *Send* and *Receive* were implemented in the *Individual* class. The former broadcasts the individual to all other demes, while the latter receives an individual from another deme and overwrites itself. The migration operation is then controlled by the *Algorithm* class, where the best individuals are sent and the worst are substituted.

## 3. Results and discussion

Two examples of design optimization of laminated structures using the discussed algorithm coupled with finite element analysis are presented. In both examples, a quadrilateral Shallow Shell element based in the Reissner–Mindlin theory with 8 nodes and 5 degrees of freedom per node is used.

In order to represent the composite material, the Classical Lamination Theory is used [15]. The shell section is thus pre-integrated in order to transform the laminate into a single equivalent ply (*Equivalent Single Layer Theory*). After the FEM analysis, the generalized stresses are then used to find the stresses and strains in the local coordinate system of each layer, which are in turn used to evaluate the failure index $\lambda_f$ of each layer using the Tsai–Wu failure criterium [18]. Additionally, the calculated stiffness matrix can be used together with the geometric stiffness matrix in a linearized stability formulation to obtain the buckling load factor $\lambda_b$.

An SGI cluster with 5 compute nodes was used to run the optimizations. The nodes are fully connected through a Gigabit network interface and each features two AMD Opteron (TM) 6234 Processors with 2.4 MHz frequency and 12 cores. A Head Node with the same configuration is also present, but is dedicated only to operating system tasks. Each node features 64 gigabytes of RAM and 3 terabytes of scratch HDD. The system runs Red Hat version 4.4.5-6. A personal desktop computer was also used to obtain speedup measures. It was equipped with a third-generation Core-i7 processor with 4 physical cores (up to 8 cores can be used through hyper-threading), 8 gigabytes of RAM and 1 terabyte of HDD running Ubuntu version 12.04.2 LTS.

### 3.1. Square plate under transverse load

The first example was taken from [2] and consists of the simultaneous weight and central deflection minimization of a square plate subjected to a uniform transverse load. The geometry, load and boundary conditions of the plate are shown in Fig. 9. It was discretized using 400 quadratic elements with reduced integration, with a total of 6405 degrees of freedom.

In this example, the concept of reliability will be used. It is defined as the number of optimizations in which at least one global optimum was found ($N_o$) divided by the total number of optimizations ($N$) [2]:

$$R(\%) = 100 \cdot \frac{N_o}{N} \tag{3}$$

In order to compare the obtained results, the plate was considered thin and thus all transverse shear stresses were neglected in the evaluation of the failure index $\lambda_f$. Likewise, the material properties

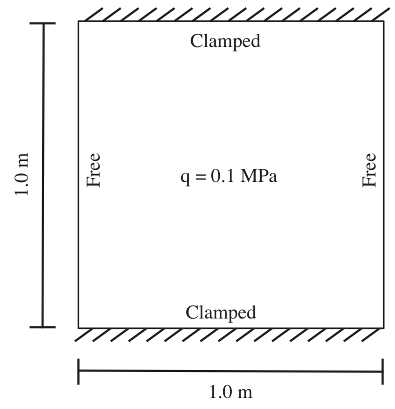


**Fig. 9.** Plate geometry and boundary conditions.

of Graphite–Epoxy plies were simplified for those of a plane stress state and are shown in Table 1.

Simultaneous weight and deflection minimization requires a multiobjective formulation, as they are opposite objectives. In [2], the Weighted Sum Method (WSM) is used to obtain the Pareto-optimal points. However, as this method cannot find points in non-convex attainable sets [19], some of the points could not be obtained. Here, the Weighted Compromise Programming Method (WCP) was used [19]. Defining $w$ as the weighting factor given to the minimization of the plate weight, the objective function for the problem can be written as:

$$f_{obj} = \left[w \cdot \frac{W - W_{min}}{W_{max} - W_{min}}\right]^m + \left[(1-w) \cdot \frac{D - D_{min}}{D_{max} - D_{min}}\right]^m \quad (4)$$

where $W$ is the weight, $D$ is the transversal displacement in the center of the plate (positive downwards) and $m$ is a power factor. The normalization limits are obtained by setting $w$ to 0 (obtaining $W_{max}$ and $D_{min}$) and to 1 (obtaining $W_{min}$ and $D_{max}$). It is important to note that if $m = 1$, the objective function is reduced to one obtained using WSM.

As for the problem constraints, the maximum failure index $\lambda_f^{max}$ is obtained by taking $\lambda_f$ in every layer of every integration point in the model. The failure constraint is thus defined as:

$$g_1 = \lambda_f^{max} - 1 \leqslant 0 \quad (5)$$

The second constraint limits the maximum total thickness of contiguous layers with the same fiber angle ($tt_{max}$) to 2 mm ($tt_{lim}$). This condition can be expressed as:

$$g_2 = \frac{tt_{max} - tt_{lim}}{tt_{lim}} \leqslant 0 \quad (6)$$

### 3.1.1. Pareto-optimal Frontier

First, the proposed algorithm will be used to find as many Pareto-optimal points as possible by changing $w$ in steps of 0.05. The possible angle and thickness values were considered the same as in [2] for the sake of comparison: $\theta \in \{-45°, 0°, 45°, 90°\}$ and $t \in \{0.75, 1.00, 1.50, 2.00\}$ mm. The laminate is also restricted to have 8 layers in a symmetric arrangement.

For each $w$, the optimization was executed 100 times with isolated subpopulations and then 20 times using 5 subpopulations with migration, also accounting for 100 subpopulations. Thus, apart from the communication time due to migration, the computational effort was the same for the two optimization sets. This arrange permits the observation of changes in the reliability ($R$) and required time per subpopulation ($T$) due to migration.

Each subpopulation has 120 individuals and is executed for 50 generations. For both sets, $r_c = 0.80$. For the isolated set, $p_m = 0.10$ and $p_s = 0.05$, while in the set with migration, $p_m \in [0.05, \ldots, 0.20]$ and $p_s \in [0.01, \ldots, 0.10]$. As the laminate is fixed at 8 layers, $p_a$ and $p_d$ are both set as zero. The results are shown in Table 2 and the Pareto frontier is shown in Fig. 10. As one of the objectives of this example is to compare the results with those obtained in [2], especially the algorithm reliability, the power factor $m$ was kept at 1 for all points except those marked with an asterisk ($*$), which were obtained by setting $m = 2$. It is important to note that all Pareto-optimal points were obtained using the WCP method, even those in the non-convex part of the frontier.

The algorithm was able to obtain global optima for every value of $w$. However, the reliability drops below 100% in some cases of the isolated set. For instance, for $w = 0.55$, $R = 74\%$ for the isolated set, whereas in the set with migration, $R = 100\%$ for all values of $w$. It is important to note that even when the reliability drops below 100%, the algorithm always finds a feasible near-optimum solution. Observing the optimization times, each subpopulation with migration takes only about 0.90% more time to run than an isolated one. This shows that the results can be improved using migration and randomized genetic operator rates with a very small impact on the total run time.

Comparing the results with those shown in [2], a lighter design was obtained for $w = 1.0$, with a reduction of 0.25 mm in the outer 90° layer. This can be attributed to differences in the finite element and Tsai–Wu formulations used, since $\lambda_f$ is very close to 1. However, when the failure constraint is not active, the present algorithm obtained the same weights as those in [2]. Comparing the plate center deflection, differences tend to appear as the plate grows thicker, with a difference of 0.50% for $w = 1.0$ and 15.6% for $w = 0$. These may also be due to differences in the used finite element formulations. Regarding the reliability, the values were generally higher even for the isolated set, with fewer points having less than 100% reliability.

### 3.1.2. Effects of migration

In the next set of optimizations, the effect of migration on the performance and reliability of the algorithm will be studied. To this end, the variable ranges were widened to augment the design space and make the solution harder, otherwise the differences in reliability would not be noticed. The lower and upper bounds of both variables were kept, but more intermediate values were added: $\theta \in [-45°, -44°, \ldots, 89°, 90°]$ and $t \in [0.75, 1.00, \ldots, 1.75, 2.00]$ mm. Additionally, in order to analyze only the effect of migration, the genetic operator rates were fixed ($r_c = 0.80$, $p_m = 0.10$, $p_s = 0.05$). To find the mean behavior of each situation, 200 subpopulations of 24 individuals were executed (40 runs with migration or 200 runs without migration), running for 300 generations.

First, the influence of the migration rate will be analyzed. Migration intervals of 1, 5, 10, 50 and 100 generations were used,

**Table 1**
Graphite–Epoxy properties.

| $E_1$ (GPa) | $E_2$ (GPa) | $G_{12}$ (GPa) | $\nu_{12}$ | $\rho$ (kN/m$^3$) | $X_T$ (MPa) | $X_C$ (MPa) | $Y_T$ (MPa) | $Y_C$ (MPa) | $S_6$ (MPa) |
|---|---|---|---|---|---|---|---|---|---|
| 181.0 | 10.3 | 7.17 | 0.28 | 15.7 | 1500 | 1500 | 40 | 246 | 68 |

**Table 2**
Results for multiple values of $w$.

| $w$ | Optimal design | $W$ (N) | $D$ (mm) | $\lambda_f$ | Isolated | | Migration | |
|---|---|---|---|---|---|---|---|---|
| | | | | | $R$(%) | $T$ (s) | $R$ (%) | $T$ (s) |
| 1.00 | $[90^{0.75}/90^{1.0}/0^{0.75}/45^{0.75}]_s$ | 102.05 | 68.8 | 0.985 | 100 | 40.91 | 100 | 41.10 |
| 0.95 | $[90^{1.0}/90^{0.75}/45^{0.75}/90^{0.75}]_s$ | 102.05 | 66.5 | 0.992 | 100 | 40.96 | 100 | 41.10 |
| 0.90 | $[90^{0.75}/90^{1.0}/45^{0.75}/90^{0.75}]_s$ | 102.05 | 66.5 | 0.992 | 100 | 40.92 | 100 | 41.17 |
| 0.85 | $[90^{0.75}/90^{1.0}/45^{0.75}/90^{0.75}]_s$ | 102.05 | 66.5 | 0.992 | 100 | 40.95 | 100 | 41.12 |
| 0.80 | $[90^{0.75}/90^{1.0}/45^{0.75}/90^{0.75}]_s$ | 102.05 | 66.5 | 0.992 | 100 | 40.97 | 100 | 41.17 |
| 0.75 | $[90^{1.0}/90^{1.0}/45^{0.75}/90^{0.75}]_s$ | 109.90 | 52.7 | 0.836 | 99 | 40.93 | 100 | 41.08 |
| 0.70 | $[90^{1.0}/90^{1.0}/45^{0.75}/90^{0.75}]_s$ | 109.90 | 52.7 | 0.836 | 100 | 40.92 | 100 | 41.18 |
| 0.65 | $[90^{1.0}/90^{1.0}/45^{0.75}/90^{1.0}]_s$ | 117.75 | 43.4 | 0.740 | 96 | 40.94 | 100 | 41.13 |
| 0.60 | $[90^{1.0}/90^{1.0}/45^{1.0}/90^{1.0}]_s$ | 125.60 | 36.3 | 0.672 | 100 | 40.95 | 100 | 41.11 |
| 0.55 | $[90^{2.0}/45^{0.75}/90^{0.75}/45^{0.75}]_s$ | 133.45 | 30.3 | 0.596 | 74 | 40.87 | 100 | 41.15 |
| 0.50 | $[90^{2.0}/45^{0.75}/90^{1.0}/45^{0.75}]_s$ | 141.30 | 25.7 | 0.536 | 96 | 40.95 | 100 | 41.11 |
| 0.45 | $[90^{2.0}/-45^{0.75}/90^{1.0}/-45^{1.0}]_s$ | 149.10 | 22.1 | 0.488 | 99 | 40.98 | 100 | 41.09 |
| 0.40 | $[90^{2.0}/45^{0.75}/90^{1.5}/45^{0.75}]_s$ | 157.00 | 18.9 | 0.440 | 100 | 40.96 | 100 | 41.16 |
| 0.35 | $[90^{2.0}/-45^{0.75}/90^{1.5}/-45^{1.0}]_s$ | 164.80 | 16.5 | 0.403 | 100 | 40.95 | 100 | 41.13 |
| 0.30 | $[90^{2.0}/45^{0.75}/90^{2.0}/45^{0.75}]_s$ | 172.70 | 14.3 | 0.336 | 99 | 40.92 | 100 | 41.11 |
| 0.25 | $[90^{2.0}/45^{0.75}/90^{2.0}/45^{1.0}]_s$ | 180.50 | 12.6 | 0.337 | 100 | 41.03 | 100 | 41.20 |
| 0.20 | $[90^{2.0}/45^{1.0}/90^{2.0}/45^{1.0}]_s$ | 188.40 | 11.4 | 0.323 | 100 | 40.96 | 100 | 41.14 |
| 0.15 | $[90^{2.0}/45^{2.0}/90^{2.0}/45^{1.0}]_s$ | 219.80 | 7.86 | 0.291 | 100 | 40.99 | 100 | 41.08 |
| 0.10 | $[90^{2.0}/45^{2.0}/90^{2.0}/45^{1.0}]_s$ | 219.80 | 7.86 | 0.291 | 100 | 40.91 | 100 | 41.09 |
| *0.10 | $[90^{2.0}/45^{1.5}/90^{2.0}/45^{0.75}]$ | 196.25 | 10.5 | 0.322 | 100 | 40.88 | 100 | 41.20 |
| *0.06 | $[90^{2.0}/45^{1.5}/90^{2.0}/45^{1.0}]_s$ | 204.1 | 9.41 | 0.306 | 100 | 41.00 | 100 | 41.18 |
| 0.05 | $[90^{2.0}/45^{2.0}/90^{2.0}/45^{1.0}]_s$ | 219.80 | 7.86 | 0.291 | 100 | 40.84 | 100 | 41.21 |
| *0.05 | $[90^{2.0}/45^{2.0}/90^{2.0}/45^{0.75}]_s$ | 211.95 | 8.71 | 0.306 | 100 | 40.85 | 100 | 41.19 |
| 0.00 | $[90^{2.0}/45^{2.0}/90^{2.0}/45^{1.0}]_s$ | 219.80 | 7.86 | 0.291 | 100 | 40.97 | 100 | 41.20 |

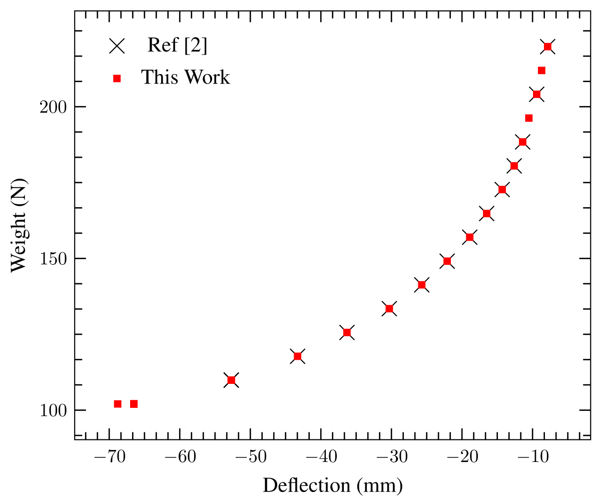


**Fig. 10.** Pareto-optimal frontier.

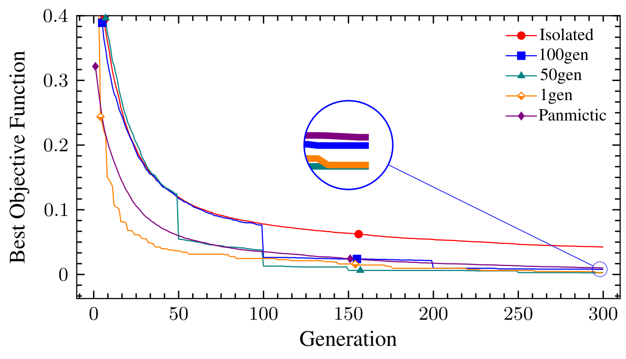


**Fig. 11.** Best individual plot with varying migration rates.

**Table 3**
Analysis of the migration rate effect.

| Migration interval | Number of migrations | Reliability ($R$) (%) | Time (s) |
|---|---|---|---|
| Isolated | 0 | 36.3 | 65.36 |
| 100 | 2 | 82.5 | 65.36 |
| 50 | 5 | 90.4 | 65.86 |
| 10 | 29 | 87.5 | 67.44 |
| 5 | 59 | 85.0 | 68.73 |
| 1 | 299 | 90.0 | 69.23 |
| Panmictic | 0 | 79.3 | 240.87 |

with the number of migrated individuals fixed to 1. Additionally, a set of isolated subpopulations was executed, as well as a panmictic set with populations of 120 individuals. The objective is to find a suitable migration rate in order to make the algorithm have the same or better efficiency of a panmictic configuration, which is much slower in a cluster with distributed memory [7].

Fig. 11 shows the best individual in each generation averaged over the 200 subpopulation runs. The graph shows the two boundary cases consisting of isolated 24-individual and 120-individual populations and also three different migration rates. The first important observation is that regardless of the migration rate, the reliability is not just equivalent to that of the panmictic population, but better. This is indicated by the average best objective function at the last generation (Fig. 11), which has a direct relation with the reliability. Those are also shown in Table 3 for all migration rates. Such behavior shows the advantage of a coarse-grain parallelization strategy, in which the optimization not only runs faster but also gives better results than large panmictic populations.

It is also interesting to observe that in the generations in which migrations occur, there is an abrupt drop in the objective function of the best individual in all subpopulations. This effectively accelerates the convergence, making it even faster than the panmictic case for high migration rates and confirms the relation of coarse-grain GAs and the *Theory of Punctuated Equilibria* [7].

Regarding the required time, as expected, the optimization time per subpopulation increases as the migration becomes more frequent, i.e. there is an increase in the communication time (Table 3). However, even with the maximum migration frequency (every generation), the total time is only about 5.9% higher than the fully isolated case. This indicates that the communication time between demes is always very small even when using a Gigabit network, as very little data is transmitted. However, as the reliability remains almost the same for all frequencies, it seems better to choose small migration rates (100–10 generations), which keeps the overhead at a minimum.

Next, the influence of the number of migrated individuals will be considered. From the aforementioned observations, a low migration rate of 100 generations was chosen, but now the number of migrated individuals was increased. The results in terms of reliability and time per subpopulation are shown in Table 4. The results suggest that both reliability and time show no sensibility to increases in the number of migrated individuals. However, as more individuals migrate, the population becomes increasingly uniform, which may bring premature convergence problems. Thus, it seems better to choose a small number of migrated individuals.

### 3.1.3. Speedup measures

The speedup was measured separately for each parallelization strategy. By definition, the speedup is the relation between the execution time of the sequential (non-parallelized) version of an algorithm and the execution time of a parallelized version with $m$ processors:

$$S_m = \frac{T_1}{T_m} \tag{7}$$

The speedup is said to be sublinear if $S_m < m$, linear if $S_m = m$ and super linear if $S_m > m$. It is important to take the speedup as an average of multiple runs, as there are time fluctuations and exogenous factors that also influence execution time. In order to measure the OpenMP speedup, a single population was executed both in a cluster node and in the personal computer. Each speedup value was an average of 5 executions. The results can be seen in Fig. 12a.

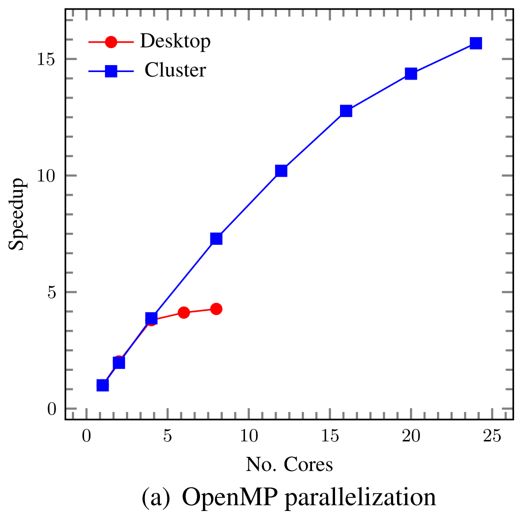
Both computers showed satisfactory speedups. It is important to note that even though 8 cores can be used in the personal computer, the performance is closer to the one with only 4 cores. This is to be expected since only 4 physical cores are present. Even though both speedups were sublinear due to the non-parallelized portions of the code, the speed gain was substantial.

Another important aspect is that a single personal computer core is about two times faster than a cluster core. However, when parallelized, the execution time clearly shows the advantage of using the cluster, as the program is executed almost two times faster. This shows the importance of parallelization in harnessing processing power, particularly with processors with many cores such as the cluster node. Such aspect is even more important in academic software, in which usually no attention is paid to parallelization.

Fig. 12b shows the speedup measures related to the MPI implementation with 5 migrations per run and an increasing number of demes. As each deme is added, the subpopulation size decreases and the algorithm runs faster. The speedup is then measured with $T_1$ being the time to run a single panmictic population. The obtained speedup is also very good, reaching 4.36 when 5 demes are used. Also, it shows an almost linear behavior, although it falls short of the theoretical value of 5 due to routines whose run times do not scale with the population size, such as input and output operations using data files.

Table 4
Analysis of the effect of the number of migrated individuals.

| Individuals per migration | Reliability ($R$) (%) | Time (s) |
|---|---|---|
| 1 | 82.5 | 65.36 |
| 2 | 87.5 | 65.36 |
| 3 | 84.0 | 65.36 |
| 4 | 84.5 | 65.36 |



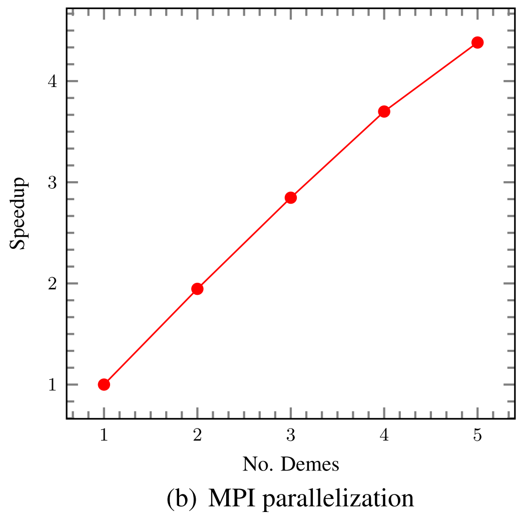


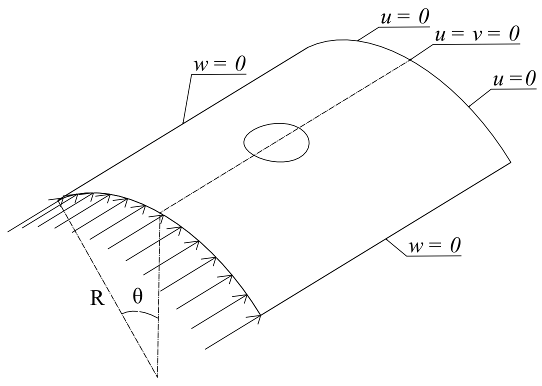
Fig. 12. Speedup measures.

### 3.2. Curved panel with circular cutout

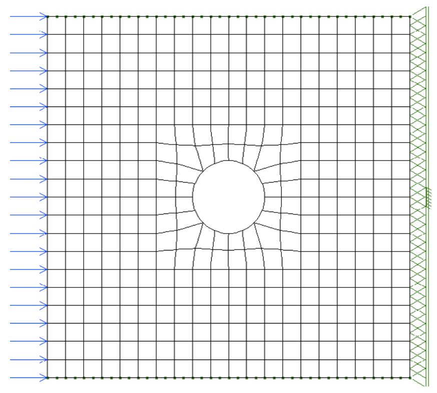
In this example, the proposed algorithm will be used to obtain optimum designs of a curved panel with a central circular cutout. Such structures are of common use, particularly in aerospace applications, featuring complex geometry, load and support conditions that cannot be analyzed using simple analytical solutions. Thus, the use of high-fidelity numerical methods such as the FEM becomes a necessity. The geometry and finite element mesh is shown in Fig. 13. A total of 384 shallow shell elements were used in order to obtain a satisfactory discretization level around the cutout, with a total of 6240 degrees of freedom.

In the $x$–$y$ plane, the shell is a square with 1 m length and the cutout radius measures 0.1 m. In the $z-y$ plane, $R = 5$ m and $\theta = 0.1$ rad. An uniformly distributed axial load is applied in one of the sides. Thus, not only its failure due to excessive axial stress but also its buckling failure have to be taken into account.

The possible design variable values were: $\theta \in \{-45°, 0°, 45°, 90°\}$ and $t \in \{0.75, 1.00, 1.25, 1.50, 1.75, 2.00\}$ mm and the number of plies was fixed at 8. Additionally, two different materials are considered and thus the material also becomes a design variable, resulting in a total of 5,308,416 possible designs. The mechanical properties of both materials are shown in Table 5. The mechanical properties of Graphite–Epoxy (ge) are clearly superior to those of Kevlar–Epoxy (ke), but it is 3 times more expensive. Consequently,



(a) Geometric Model



(b) Finite Element Mesh

**Fig. 13.** Curved panel example.

the algorithm has to balance the cost and structural performance of the laminate.

The objective is to minimize material cost ($C_t$), which is calculated using the total weight of each material in the laminate:

$$C_t = W_{ke} \cdot C_{ke} + W_{ge} \cdot C_{ge} \tag{8}$$

where $W$ is the weight for each material and $C$ is the material relative cost per unit weight showed in Table 5. The stress failure and contiguous ply thickness constraints used in the previous example are applied here, and a buckling constraint, represented by the factor $\lambda_b$, is also used. Such factor represents the critical buckling load relative to the applied load. The constraint expressions are given by:

$$\begin{aligned} g_1 &= \lambda_f^{max} - 1 \leqslant 0 \\ g_2 &= \frac{tt_{max} - tt_{lim}}{tt_{lim}} \leqslant 0 \\ g_3 &= 1 - \lambda_b \leqslant 0 \end{aligned} \tag{9}$$

**Table 5**
Material properties [2].

| | Kevlar–Epoxy | Graphite–Epoxy |
|---|---|---|
| $E_1$ (GPa) | 87 | 181 |
| $E_2 = E_3$ (GPa) | 5.5 | 10.3 |
| $\nu_{12}$ | 0.34 | 0.28 |
| $\nu_{13}$ | 0.30 | 0.30 |
| $\nu_{23}$ | 0.49 | 0.49 |
| $G_{12} = G_{13}$ (GPa) | 2.2 | 7.17 |
| $G_{23}$ (GPa) | 1.47 | 4.80 |
| $X_t$ (MPa) | 1280 | 1500 |
| $X_c$ (MPa) | 335 | 1500 |
| $Y_t = Z_t$ (MPa) | 30 | 40 |
| $Y_c = Z_c$ (MPa) | 158 | 246 |
| $S_6 = S_5 = S_4$ (MPa) | 49 | 68 |
| Cost (units/N) | 1.0 | 3.0 |

**Table 6**
Influence of subpopulation size and number of generations.

| | 24 ind | 48 ind | 72 ind | 96 ind |
|---|---|---|---|---|
| 50 gen | 31 (0.02) | 43 (0.04) | 46 (0.05) | 72 (0.07) |
| 100 gen | 60 (0.03) | 70 (0.07) | 86 (0.11) | 87 (0.14) |
| 150 gen | 70 (0.05) | 82 (0.11) | 83 (0.16) | 90 (0.22) |
| 200 gen | 80 (0.07) | **95** (**0.14**) | 94 (0.21) | 98 (0.29) |

Five demes were used and migration was maintained fixed at one individual every 10 generations. The genetic parameters were $r_c = 0.80$, $p_m \in [0.05, \ldots, 0.20]$, $p_s \in [0.01, \ldots, 0.05]$, $p_a \in [0.01, \ldots, 0.05]$ and $p_d \in [0.01, \ldots, 0.05]$. First, the applied load was fixed at 400 kN/m and the effect of the subpopulation size and generation number in the reliability ($R$) was studied. For each case, 200 subpopulations were executed and the average reliability was obtained. Also, the number of objective function evaluations was tracked and expressed as a percentage of the total design space (%DS). This can be understood as another measure of the algorithm performance and also an indirect measure of the total optimization time. Four subpopulation sizes and four values for the number of generations were combined and the results are shown in Table 6, where the first number is the reliability $R$ (%) and the number inside the parenthesis is the explored percentage of the design space %DS. Even though low reliability values were obtained in some cases, all final designs were feasible, albeit not a global optimum.

The chosen combination is expressed in boldface in Table 6, which presented a high reliability with a relatively low number of individual evaluations. Both the number of generations and the subpopulation size affect the algorithm reliability but in this particular problem the number of generations seems to have a bigger impact in improving reliability than the subpopulation size. Also, due to the random behavior of the GA, coupled with the use of random operator rates, the reliability appears to get worse when using 72 individuals and 100 and 150 generations.

Next, the applied load was varied in order to investigate the choice of materials. The load ranged from 50 kN/m to 450 kN/m and the results are shown in Table 7. As expected, the total laminate cost increases as the load becomes higher, partly because of the higher total thickness, but also because of a gradual change of materials. For lower loads, the Kevlar–Epoxy is used in all layers due to its lower price. However, as the load increases, there is an increasing material shift in the outer layers. This occurs due to the bending dominated nature of the problem, which is evidenced by the active buckling constraint. In such problems, the inner layers, being closer to the middle surface, have a small impact on the structural response. This is also evidenced by the inactive failure constraint, which is linked to the laminate membrane behavior.

**Table 7**
Results for multiple load values.

| Load (kN/m) | Optimum design | %KE | %GE | Cost | $\lambda_f$ | $\lambda_b$ | $t$(mm) |
|---|---|---|---|---|---|---|---|
| 50 | $\left[0_{ke}^{0.75}/45_{ke}^{0.75}/-45_{ke}^{1.0}/0_{ke}^{1.0}\right]s$ | 100 | 0 | 91.53 | 0.128 | 1.003 | 7.0 |
| 100 | $\left[45_{ke}^{0.75}/-45_{ke}^{1.25}/0_{ke}^{2.0}/90_{ke}^{0.75}\right]s$ | 100 | 0 | 124.22 | 0.171 | 1.006 | 9.5 |
| 200 | $\left[-45_{ke}^{1.75}/45_{ke}^{2.0}/90_{ke}^{2.0}/0_{ke}^{1.0}\right]s$ | 100 | 0 | 176.52 | 0.368 | 1.008 | 13.5 |
| 300 | $\left[-45_{ke}^{2.0}/45_{ge}^{2.0}/0_{ke}^{2.0}/45_{ke}^{1.0}\right]s$ | 71.4 | 28.6 | 313.24 | 0.464 | 1.016 | 14.0 |
| 400 | $\left[-45_{ge}^{1.75}/45_{ge}^{2.0}/0_{ke}^{2.0}/90_{ke}^{1.0}\right]s$ | 44.4 | 55.6 | 420.61 | 0.453 | 1.005 | 13.5 |
| 450 | $\left[-45_{ge}^{1.75}/45_{ge}^{2.0}/0_{ge}^{2.0}/90_{ke}^{1.0}/\right]s$ | 14.8 | 85.2 | 550.78 | 0.172 | 1.008 | 13.5 |

## 4. Conclusion

This paper presented a hybrid shared/distributed memory parallel genetic algorithm formulation to deal with laminated composite structural optimization. The algorithm was then used in conjunction with a shallow shell finite element to find optimal solutions of a plate with simple boundary conditions and a curved panel with a circular cutout.

To speed up the algorithm, two different parallelization strategies were mixed, the Global Parallelization and the Coarse-Grain techniques. The resultant hybrid parallel GA makes use of the best features of both OpenMP and MPI and can run in personal computers and high-performance cluster computers alike. Also, as a multideme configuration was used, different genetic parameters were applied in each deme in order to consider multiple evolution environments concurrently and avoid the need of tuning them for each optimization problem.

In the first example, the effects of migration in both the execution time and the algorithm reliability were studied. It was found that an island-model GA not only makes the algorithm run faster, but also improves the quality of the results when comparing a single population with a set of demes of the same total number of individuals. Also, good speedups were obtained in both parallelization levels. In the second example, a curved panel with a central circular cutout was considered. It was shown that the proposed algorithm is able to deal with complex FEM analysis procedures while maintaining relatively low execution times due to parallelization.

The proposed algorithm performed well in both examples, with high reliability and low execution times. The speed gains due to parallelization are particularly important as they permit the use of complex analysis procedures and a higher number of design variables, allowing the use of GAs in the solution of practical laminate design problems.

## Acknowledgements

The financial support by CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) and CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) is gratefully acknowledged.

## References

[1] Gurdal Z, Haftka RT, Hajela P. Design and optimization of laminated composite materials. John Wiley & Sons; 1999.
[2] Almeida FS, Awruch AM. Design optimization of composite laminated structures using genetic algorithms and finite element analysis. Compos Struct 2009;88:443–54.
[3] Park JH, Hwang JH, Lee CS, Hwang W. Stacking sequence design of composite laminates for maximumstrength using genetic algorithms. Compos Struct 2001;52:217–31.
[4] Goldberg DE. Genetic algorithms in search, optimization and machine learning. Addison-Wesley; 1989.
[5] Lima BSLP, Jacob BP, Ebecken NFF. A hybrid fuzzy/genetic algorithm for the design of offshore oil production risers. Int J Numer Methods Eng 2005;64:1459–82.
[6] Yang H, Jiang R, Li H. Optimization design of deepwater steel catenary risers using genetic algorithm. Comput Struct Eng 2009:901–8.
[7] Cantu-Paz E. A survey of parallel genetic algorithms. Tech. rep.; 1997.
[8] Punch III WF, Averill RC, Goodman ED, Lin S-C, Ding Y, Yip YC. Optimal design of laminated composite structures using coarse-grain parallel genetic algorithms. Comput Syst Eng 1994;5:415–23.
[9] Omkar SN, Venkatesh A, Mudigere M. Mpi-based parallel synchronous vector evaluated particle swarm optimization for multi-objective design optimization of composite structures. Eng Appl Artif Intell 2012;25:1611–27.
[10] OpenMP Architecture Review Board. OpenMP application program interface version 3.1; 2011.
[11] Gabriel E, Fagg GE, Bosilca G, Angskun T, Dongarra JJ, Squyres JM, et al. Open MPI: goals, concept, and design of a next generation MPI implementation. In: Proceedings, 11th European PVM/MPI Users' Group Meeting. Budapest, Hungary; 2004. p. 97–104.
[12] Barbosa HJC, Lemonge ACC. An adaptive penalty scheme for genetic algorithms in structural optimization. Int J Numer Methods Eng 2004;59:703–36.
[13] Henderson JL. Laminated plate design using genetic algorithms and parallel processing. Comput Syst Eng 1994;5:441–53.
[14] Rahul, Chakraborty D, Dutta A. Optimization of FRP composites against impact induced failure using island model parallel genetic algorithm. Compos Sci Technol 2005;65:2003–13.
[15] Reddy JN. Mechanics of laminated composite plates and shells: theory and analysis. 2nd ed. CRC Press; 2004.
[16] Alba E, Troya JM. A survey of parallel distributed genetic algorithms. Complexity 1999;4:31–52.
[17] Tanese R. Parallel genetic algorithms for a hypercube. In: Proceedings of the second international conference on genetic algorithms on genetic algorithms and their application. Hillsdale, NJ, USA: L. Erlbaum Associates Inc.; 1987. p. 177–83.
[18] Daniel IM, Ishai O. Engineering mechanics of composite materials. 2nd ed. Oxford University Press; 2006.
[19] Athan TW, Papalambros PY. A note on weighted criteria methods for compromise solutions in multi-objective optimization. Eng Optim 1996;27:155–76.