electronics

MDPI

Article

Efficient Subpopulation Based Parallel TLBO Optimization Algorithms

Alejandro García-Monzó [1], Héctor Migallón [1,*], Antonio Jimeno-Morenilla [2], José-Luis Sánchez-Romero [2], Héctor Rico [2] and Ravipudi Venkata Rao [3]

[1] Department of Physics and Computer Architecture, Miguel Hernández University, E-03202 Alicante, Spain; garciamonzo.alejandro@gmail.com
[2] Department of Computer Technology, University of Alicante, E-03071 Alicante, Spain; jimeno@dtic.ua.es (A.J.-M.); sanchez@dtic.ua.es (J.-L.S.-R.); hector.rico@gmail.com (H.R.)
[3] Sardar Vallabhbhai National Institute of Technology, Surat 395 007, Gujarat State, India; ravipudirao@gmail.com
* Correspondence: hmigallon@umh.es; Tel.: +34-966658390

Received: 27 November 2018; Accepted: 21 December 2018; Published: 23 December 2018

Abstract: A numerous group of optimization algorithms based on heuristic techniques have been proposed in recent years. Most of them are based on phenomena in nature and require the correct tuning of some parameters, which are specific to the algorithm. Heuristic algorithms allow problems to be solved more quickly than deterministic methods. The computational time required to obtain the optimum (or near optimum) value of a cost function is a critical aspect of scientific applications in countless fields of knowledge. Therefore, we proposed efficient algorithms parallel to Teaching-learning-based optimization algorithms. TLBO is efficient and free from specific parameters to be tuned. The parallel proposals were designed with two levels of parallelization, one for shared memory platforms and the other for distributed memory platforms, obtaining good parallel performance in both types of parallel architectures and on heterogeneous memory parallel platforms.

Keywords: TLBO; optimization problems; parallel; heuristic; subpopulations; OpenMP; MPI; hybrid MPI/OpenMP

1. Introduction

The purpose of optimization algorithms is to find the optimal value for a particular cost function. Cost functions, depending on the application in which they are used, can be highly complex, it may be necessary to repeatedly obtain a new optimum value, and they may present different numbers of parameters (or design variables). Moreover, if cost functions have local minimums, the search for the optimum value becomes more complicated.

When deterministic methods have been applied to obtain the optimal value of a function, a sequence of points tending to the global optimum value is generated considering the analytical properties of the problem under consideration. In other words, the search for the optimum is treated as a problem of linear algebra, often based on the gradient of the function. The optimal value, or a value very close to it, of a cost function can be obtained using deterministic methods (see [1]). In some cases, however, the efforts involved can be considerable, for example in non-convex or large-scale optimization problems. When deterministic methods can be applied, the results obtained are unequivocable and replicable, but the computational cost can make it useless. Several heuristic methods have been proposed to address these drawbacks, many of them based on phenomena found in nature, leading to acceptable solutions while reducing the required efforts. Two main

groups of this type of algorithm, evolutionary algorithms and swarm intelligence, include the major heuristic algorithms.

On the one hand, metaheuristic methods are able to accelerate convergence even when local minima exist, and on the other, they can be used in functions whose characteristics prevent the use of deterministic methods, for example non-differentiable functions. In most cases, metaheuristic methods employ guided search techniques, in which some random processes are involved to solve the problem, although it cannot be formally proven that the optimal value obtained is the solution to the problem. In particular, the Teaching-learning-based optimization (TLBO) algorithm, presented in [2], has proven its effectiveness in a wide range of applications. For example in [3], it is used for the optimal coordination of directional overcurrent relays in a looped power system; in [4], a multi-objective TLBO is used to solve the optimal location of automatic voltage regulators in distribution systems in the presence of distributed generators; in [5], an improved multi-objective TLBO is applied to optimize an assembly line to produce large-sized high-volume products such as cars, trucks and engineering machinery; in [6], a load shedding algorithm for alleviating line overloads employs a TLBO algorithm; in [7], a TLBO algorithm is used to optimize feedback gains and the switching vector of an output feedback sliding mode controller for a multi area multi-source interconnected power system; in [8], the TLBO method is used to train and accelerate the learning rate of a model designed to forecast both wind power generation in Ireland and that of a single wind farm, in order to demonstrate the effectiveness of the proposed method; in [9] Cetane number estimation of biodiesel with a fatty acid methyl esters composition was performed using a hybrid optimization method including a TLBO algorithm; in [10], a residential demand side management scheme based on electricity cost and peak to average ratio alleviation with maximum user satisfaction is proposed using a hybrid technique based on TLBO and enhanced differential evolution (EDE) algorithms; in [11] a TLBO algorithm is used in Transmission Expansion Planning (TEP) that involves determining if and how transmission lines should be added to the power grid, considering power generation costs, power loss, and line construction costs among others.

Among the well-known metaheuristic optimization algorithms based on natural phenomena, it is worth mentioning: Particle Swarm Optimization (PSO) and its variants, Artificial Bee Colony (ABC), Shuffled Frog Leaping (SFL), Ant Colony Optimization (ACO), Evolutionary Strategy (ES), Evolutionary Programming (EP), Genetic Programming (GP), the Fire Fly (FF) algorithm, the Gravitational Search Algorithm (GSA), Biogeography-Based Optimization (BBO), the Grenade Explosion Method (GEM), Genetic Algorithms (GA) and its variants, Differential Evolution (DE) and its variants, Simulated Annealing (SA) algorithm and the Tabu Search (TS) algorithm can be mentioned.

In most of these algorithms, it is necessary to adjust one or more parameters first, for example, GA needs crossover probability, mutation probability, selection operator, etc. to be set correctly; the SA algorithm needs the initial annealing temperature and cooling schedule to be tuned; PSO's specific parameters are inertia weight and social and cognitive parameters; HSA needs the harmony memory consideration rate, the number of improvisations, etc. to be set correctly; and the immigration rate, emigration rate, etc., need to be tuned for BBO. The population-based heuristic algorithm used in this work, the Teacher-Learner Based Optimization (TLBO) [12] overcomes the problem of tuning algorithm-specific parameters. Specifically, the TLBO algorithm only needs general parameters to be set, such as the number of iterations, population size and stopping criterion.

Some recent works applied TLBO algorithm parallelization techniques. For example, authors in [13] implemented a TLBO algorithm on a multicore processor within an OpenMP environment. The OpenMP strategy emulated the sequential TLBO algorithm exactly, so the calculation of fitness, calculation of mean, calculation of best, and comparison of fitness functions remained the same, while small changes were introduced to achieve better results. A set of 10 test functions were evaluated when running the algorithm on a single core architecture, and were then compared on architectures ranging from 2 to 32 cores. Average speed-up values of $4.9x$ and $6.4x$ with 16 and 32 processors were obtained respectively, corresponding to efficiencies of 30% and 20% respectively. In [14], the authors

propose a parallel TLBO procedure for automatic heliostat aiming, obtaining good speed-up values for this extremely expensive problem using up to 32 processes; parallel performance, however, worsened when using functions that were not so computationally expensive.

Other parallel proposals for different heuristic optimization algorithms have been proposed. For example, authors in [15] implemented the Dual Population Genetic Algorithm (DPGA) on a parallel architecture obtaining average speed-up values of $1.64x$ using both 16 and 32 processors. The authors in [16] propose a parallel version of the ACO metaheuristic algorithm obtaining a maximum speed-up of $5.94x$ using 8 processors, going down to $5.45x$ when using 16 processors. In addition, other proposals use hardware accelerators. For example, in [17], the PSO algorithm is accelerated using FPGAs and in [18], the Jaya algorithm is accelerated through the use of GPUs.

In Section 2, we present the TLBO optimization algorithm and describe the parallel algorithms in Section 3. In Section 4, we analyse the latter in terms of parallel performance and optimization behaviour, and some conclusions are drawn in Section 5.

2. The TLBO Algorithm

The Teaching-Learning-Based Optimization (TLBO) algorithm, like all evolutionary and swarm intelligence-based algorithms, requires common controlling parameters, but does not require algorithm-specific control parameters. Both these algorithms and TLBO are population-based and probabilistic algorithms, therefore TLBO needs to set only the size of the populations and number of generations.

The TLBO algorithm is based on common teaching and learning processes of a group of students, whose learning process is influenced both by the teacher and by interactions within the group of students. Each source of advancement of knowledge (that allows to approach the solution to the problem) is associated with a different phase of the TLBO algorithm, the first phase is the teacher phase and the second is the learner phase.

As mentioned previously, the TLBO is a population-based heuristic algorithm, therefore the first step is the creation of the initial population (line 1 of Algorithm 1). A population is a set of m individuals; each individual is composed of k variables (design variables) and the value of k depends on the cost function (F_{cost}) to be optimized. Each individual in the initial population is created as shown in Equation (1), where $r_{i,j}$ are uniformely distributed random numbers, and $minVar_j$ and $maxVar_j$ specify the domain size of each variable.

$$X_{i,j} = minVar_j + (maxVar_j - minVar_j) * r_{i,j} \tag{1}$$

Once the population is created, the teacher phase begins by identifying the individual that will act as teacher (line 6 of Algorithm 1). The teacher will be the individual possessing the greatest amount of knowledge, i.e., the individual whose solution is the best among all individuals in the population. In the learner phase, the teacher tries to improve students' knowledge. To model this interaction, the mean of each design variable (M_j) is calculated considering all individuals in the population, and the interaction is performed considering the mean values computed: the teacher ($X_{teacher}$), the teaching factor (TF), as well as a random factor (r_j). The teaching factor is an integer random value in the range of $[1, 2]$, while the random factor is a random real value in the range of $[0, 1]$. In other words, the teaching factor is an integer value equal to 1 or equal to 2 that is randomly chosen for each teacher phase, i.e., teaching factor is not a parameter to be tuned. While r_j are k floating-point random numbers uniformely distributed between 0 and 1.

Each individual is influenced by the teacher (line 12 of Algorithm 1). If the influence is positive, i.e., if it improves the student, the new student replaces the previous student in the population. Whorthy of note, in line 14 of Algorithm 1 a minimization problem is considered. The resulting population at the end of the teacher phase will be the initial population used in the learner phase ($Y_{i,j}$ in Algorithm 2).

Algorithm 1 Teacher phase of TLBO algorithm.

Create Initial Population: $X_{i,j}$
  i identifies the individual $i = 1 \ldots m$
  j identifies the design variable $j = 1 \ldots k$
Teacher phase:
{
Identify the best individual or teacher ($X_{teacher}$)
Compute the mean of all design variables M_j
Compute the teaching factor (TF)
Compute the random factors (r_j)
for $i = 1$ to m **do**
  for $j = 1$ to k **do**
    $X'_{i,j} = X_{i,j} + r_j(X_{teacher,j} - TF \times M_j)$
  end for
  if $F_{cost}(X'_i) < F_{cost}(X_i)$ **then**
    Replace X_i by X'_i
  end if
end for
}

Algorithm 2 Learner phase of TLBO algorithm.

Initial population in the learner phase: $Y_{i,j}$
  i identifies the individual $i = 1 \ldots m$
  j identifies the design variable $j = 1 \ldots k$
Learner phase:
{
for $i = 1$ to m **do**
  Randomly identify another student with whom to interact (p)
  if $F_{cost}(Y_i) < F_{cost}(Y_p)$ **then**
    for $j = 1$ to k **do**
      $Y'_{i,j} = Y_{i,j} + r_{i,j}(Y_{i,j} - Y_{p,j})$
    end for
  else
    for $j = 1$ to k **do**
      $Y'_{i,j} = Y_{i,j} + r_{i,j}(Y_{p,j} - Y_{i,j})$
    end for
  end if
  if $F_{cost}(Y'_i) < F_{cost}(Y_i)$ **then**
    $Z_i = Y'_i$
  else
    $Z_i = Y_i$
  end if
end for
Output population in learner phase: $Z_{i,j}$
  i identifies the individual $i = 1 \ldots m$
  j identifies the design variable $j = 1 \ldots k$
}

In the second stage, the learner phase, the students' knowledge can improve due to the influence of the students themselves, i.e., by the interaction between them. In the learner phase, shown in Algorithm 2, each student (or individual) interacts with another student, who is randomly chosen. Worthy of note, the initial population ($Y_{i,j}$) is the resulting population at the end of the teacher phase.

Once both students are identified the interaction between them depends on the most learned student, i.e., it depends on the evaluation of the cost function for the two interacting students (lines 8–16 of Algorithm 2). The result of this interaction is an individual who is evaluated and compared with the initial individual, so the best among them is transferred to the population resulting from the learner phase ($Z_{i,j}$). Worthy of note, in the teacher phase algorithm, a minimization problem is considered in line 17 of Algorithm 2.

The teacher and learner phases are repeated until the stop criterion is met. The number of repetitions (determined by the "Iterations" parameter) specifies the number of generations to be created. Significantly, the resulting population of the learner phase ($Z_{i,j}$) is the initial population for the teacher phase in the next iteration. All random numbers used in Algorithms 1 and 2 (r_j and $r_{i,j}$) are uniformely distributed random numbers in the range of $[0, 1]$.

3. Parallel Approaches

We propose hybrid OpenMP/MPI parallel algorithms to exploit heterogeneous memory platforms. The whole sequential TLBO algorithm is shown in Algorithm 3. The "Runs" parameter corresponds to the number of independent executions performed. Therefore, in line 21 of Algorithm 3, "Runs" different solutions should be evaluated. In each independent execution both teacher and learner phases are repeated "Iterations" times. The parallel approach to exploit distributed memory platforms is applied to independent executions (line 5 of Algorithm 3), while the parallel approaches to exploit shared memory platforms are applied using subpopulations in teacher and learner phases as well as in the duplicate removal phase. The elimination of duplicates is necessary to avoid premature convergence.

Algorithm 3 Skeleton of the sequential TLBO algorithm.

```
1:  Define function to minimize (F_cost)
2:  Set Runs parameter
3:  Set Iterations parameter
4:  Set m parameter (used in Algorithms 1 and 2
5:  for l = 1 to Runs do
6:     Create New Population (Z_1):
7:     for q = 1 to Iterations do
8:        Teacher phase:
9:          (Input: Population Z_q)
10:         (Output: Population Y_q)
11:       Learner phase:
12:         (Input: Population Y_q)
13:         (Output: Population Z'_{q+1})
14:       Duplicate removal phase:
15:         (Input: Population Z'_{q+1})
16:         (Output: Population Z_{q+1})
17:    end for
18:    Store Solution
19:    Delete Population
20: end for
21: Obtain Best Solution and Statistical Data
```

We developed two parallel proposals in order to exploit shared memory platforms. Both proposals distribute the work load associated with teacher and learner phases by considering subpopulations. The size of the whole population is equal to m; if the number of parallel threads (or processes) is nt, we consider nt subpopulations of sizes m_{nt}, where $\sum m_{nt} = m$. In the first proposal, called **SPG_ParTLBO**, the whole population is partitioned into subpopulations (SP) that are stored in global (G) memory. While in the second proposal, called **SPP_ParTLBO**, the whole population is also partitioned into subpopulations (SP), but they are stored in private (P) memory.

Algorithm 4 shows the parallel teacher phase for the SPG_ParTLBO algorithm. In line 5 all threads compute the initial subpopulation and store it in global memory; in line 12, the best individual of each subpopulation is identified, and the teacher (the global best individual) is sequentially identified in line 14. Following a similar strategy, the means of the design variables of each subpopulation are calculated in line 16, and in line 18 the global value of these mean values are obtained sequentially. Finally, the influence of the teacher is applied to each individual in parallel, introducing those who have improved their knowledge into the population (line 27). The parallel teacher phase shown in Algorithm 4 does not modify the optimization procedure of the sequential algorithm shown in Algorithm 1.

Algorithm 4 Teacher phase of SPG_TLBO algorithm.

1: Set population size parameter (m)
2: Obtain the number of parallel threads (nt)
3: Compute the size of subpopulations (m_{nt})
4: **In parallel** $s = 1$ to nt **do**
5: Create Initial Subpopulation: X_{i_s}
6: **end for**
7: {Whole Population is: $X_{i,j}$ }
8: **Teacher phase:**
9: {
10: Compute the teaching factor (TF)
11: **In parallel** $s = 1$ to nt **do**
12: Identify the best individual of subpopulation (X_{best_s})
13: **end for**
14: Compute the global teacher: $X_{teacher} = Bestof(X_{best_s})$
15: **In parallel** $s = 1$ to nt **do**
16: Compute the partial mean of all design variables M_{j_s}
17: **end for**
18: Compute the global mean of all design variables M_j
19: **In parallel** $s = 1$ to nt **do**
20: Compute the random factors (r_{j_s})
21: **end for**
22: **In parallel** $s = 1$ to nt **do**
23: **for** $i = 1$ to m_{nt} **do**
24: **for** $j = 1$ to k **do**
25: $X'_{i_s,j} = X_{i_s,j} + r_{j_s}(X_{teacher,j} - TF \times M_j)$
26: **end for**
27: **if** $F_{cost}(X'_{i_s}) < F_{cost}(X_{i_s})$ **then**
28: Replace X_{i_s} by X'_{i_s}
29: **end if**
30: **end for**
31: **end for**
32: }

Algorithm 5 shows the parallel learner phase for the SPG_ParTLBO algorithm. Each process, for each student in its subpopulation, randomly chooses another student with whom to interact, who can be located in any subpopulation since the whole population is stored in global memory (line 5). The rest of the code (lines 6–20) remains unchanged with respect to the sequential algorithm shown in Algorithm 2.

Algorithm 5 Learner phase of SPG_TLBO algorithm.

```
1: Learner phase:
2: {
3: In parallel s = 1 to nt do
4:   for i = 1 to m_nt do
5:     Randomly identify another student with whom to interact (p ∈ [1, m])
6:     if F_cost(Y_i) < F_cost(Y_p) then
7:       for j = 1 to k do
8:         Y'_{i,j} = Y_{i,j} + r_{i,j}(Y_{i,j} − Y_{p,j})
9:       end for
10:    else
11:      for j = 1 to k do
12:        Y'_{i,j} = Y_{i,j} + r_{i,j}(Y_{p,j} − Y_{i,j})
13:      end for
14:    end if
15:    if F_cost(Y'_i) < F_cost(Y_i) then
16:      Z'_i = Y'_i
17:    else
18:      Z'_i = Y_i
19:    end if
20:  end for
21: end for
22: }
```

The duplicate removal phase for the SPG_ParTLBO algorithm, shown in Algorithm 6, performs the same procedure as the sequential procedure in parallel. Worthy of note, when a duplicate is found, a random design variable is chosen to be modified.

Algorithm 6 Duplicate removal phase of SPG_TLBO algorithm.

```
1: Duplicate removal phase:
2: {
3: In parallel s = 1 to nt do
4:   for i ∈ m_nt do
5:     for j = i + 1 to m do
6:       if Z'_i = Z_j then
7:         Select randomly one design variable (s ∈ [1, k])
8:         Randomly change Z'_{i_s}
9:       end if
10:    end for
11:  end for
12: end for
13: Z = Z'
14: }
```

To increase parallel efficiency, we developed the second proposal called SPP_ParTLBO, in which subpopulations are stored in private memory at each thread. However, the subpopulations are not isolated structures. Algorithm 7 shows the parallel learner phase for the SPP_ParTLBO algorithm. As can be seen, after identifying the best individual (i.e., the teacher) the thread that stored it in its subpopulation copies it into the global memory, so all the threads use the same teacher (lines 11–17).

In contrast, the means of the design variables used in each subpopulation are obtained only with the individuals of the subpopulation (line 20). The rest of the teacher phase remains unchanged.

Algorithm 7 Teacher phase of SPP_TLBO algorithm.

1: Set population size parameter (m)
2: Obtain the number of parallel threads (nt)
3: Compute the size of subpopulations (m_{nt})
4: **In parallel** $s = 1$ to nt **do**
5: Create Initial Subpopulation: X_{i_s}
6: **end for**
7: {Whole Population is: $X_{i,j}$ }
8: **Teacher phase:**
9: {
10: Compute the teaching factor (TF)
11: **In parallel** $s = 1$ to nt **do**
12: Identify the best individual of subpopulation (X_{best_s})
13: **end for**
14: Calculate the best global individual (teacher) and its owner thread
15: **if** Is the owner of the teacher **then**
16: Copy teacher to global memory $Best_{global}$
17: **end if**
18: Sync BARRIER
19: **In parallel** $s = 1$ to nt **do**
20: Compute the mean of design variables in subpopulation M_{j_s}
21: **end for**
22: **In parallel** $s = 1$ to nt **do**
23: Compute the random factors (r_{j_s})
24: **end for**
25: **In parallel** $s = 1$ to nt **do**
26: **for** $i = 1$ to m_{nt} **do**
27: **for** $j = 1$ to k **do**
28: $X'_{i_s,j} = X_{i_s,j} + r_{j_s}(Best_{global,j} - TF \times M_{j_s})$
29: **end for**
30: **if** $F_{cost}(X'_{i_s}) < F_{cost}(X_{i_s})$ **then**
31: Replace X_{i_s} by X'_{i_s}
32: **end if**
33: **end for**
34: **end for**
35: }

Parallel learner phase for the SPP_ParTLBO algorithm is shown in Algorithm 8. In this algorithm, the student's search range with which each student interacts is restricted to the subpopulation, not to the entire population (line 5), while the rest of the teacher' phase remains unchanged.

Algorithm 8 Learner phase of SPP_TLBO algorithm.

Learner phase:
{
In parallel $s = 1$ to nt **do**
  for $i = 1$ to m_{nt} **do**
    Randomly identify another student with whom to interact ($p \in [1, m_{nt}]$)
    if $F_{cost}(Y_i) < F_{cost}(Y_p)$ **then**
      for $j = 1$ to k **do**
        $Y'_{i,j} = Y_{i,j} + r_{i,j}(Y_{i,j} - Y_{p,j})$
      end for
    else
      for $j = 1$ to k **do**
        $Y'_{i,j} = Y_{i,j} + r_{i,j}(Y_{p,j} - Y_{i,j})$
      end for
    end if
    if $F_{cost}(Y'_i) < F_{cost}(Y_i)$ **then**
      $Z_i = Y'_i$
    else
      $Z_i = Y_i$
    end if
  end for
end for
}

In the SPP_ParTLBO algorithm, the duplicate removal phase shown in Algorithm 9, changes with respect to the sequential procedure, by restricting the search to the subpopulation, which is stored in private memory.

Algorithm 9 Duplicate removal phase of SPP_TLBO algorithm.

Duplicate removal phase:
{
In parallel $s = 1$ to nt **do**
  for $i = 1$ to m_{nt} **do**
    for $j = i + 1$ to m_{nt} **do**
      if $Z'_i = Z_j$ **then**
        Select randomly one design variable ($s \in [1, k]$)
        Randomly change Z'_{i_s}
      end if
    end for
  end for
end for
$Z = Z'$
}

To use heterogeneous memory platforms (clusters) we need to develop a hybrid memory model algorithm. As explained in Section 2, and as can be seen in Algorithm 3, the TLBO algorithm performs several fully independent executions ("Runs"). Therefore, we developed a parallel algorithm, at a higher level, for distributed memory platforms, load balance being a key aspect. The high level parallel algorithm needed to include load balance mechanisms and be able to include parallel algorithms previously described, developed for shared memory platforms.

The high level parallel TLBO algorithm focuses on the fact that all iterations in line 5 in Algorithm 3 are actually independent executions. Therefore, the total number of executions ("Runs") to be performed is divided among **np** available processes, taking into account that it cannot be distributed statically. The high level parallel algorithm must be designed for distributed memory platforms using MPI. On the one hand, we must develop a load balance procedure, and on the other, a final data gathering process (data collection from all processes) must be performed.

The developed hybrid MPI/OpenMP algorithm is shown in Algorithm 10. In this algorithm, if the number of desired worker processes is equal to np, the total number of distributed memory processes will be $np + 1$. This is because a critical process (distributed memory process) will be in charge of distributing the computing work among the np available working processes. We call this process the work dispatcher. Although the work dispatcher process is critical, it will be running in one of the nodes with worker processes, because no significant overhead is introduced in the overall parallel algorithm performance. The work dispatcher will be waiting to receive a work request signal from an idle worker process. When a particular worker process requests new work (independent execution), the dispatcher will assign a new independent execution or send an end of work signal.

Algorithm 10 Heterogeneous memory parallel TLBO algorithm.

```
1:  np: number of distributed memory worker processes
2:  Dispatcher process:
3:  {
4:  for l = 1 to Runs do
5:      Receive idle signal
6:      Send work signal
7:  end for
8:  for l = 1 to np do
9:      Receive idle signal
10:     Send end of work signal
11: end for
12: }
13: Worker processes:
14: {
15: while true do
16:     Send idle signal
17:     if end of work signal then
18:         Break while
19:     else
20:         nt: number threads
21:         Compute 1 run of SPG_TLBO or SPP_TLBO algorithm
22:         Store Solution
23:     end if
24: end while
25: }
26: Collect all the solutions and obtain Best Solution
```

The computational load of the dispatcher process is negligible, as can be observed in lines 4 to 11 of Algorithm 10. In line 21 one of the two parallel proposals of the TLBO algorithm is used, i.e., SPG_TLBO or SPP_TLBO. The total number of processes is equal to $tp = np * nt$, where np is the number of distributed memory worker processes (MPI processes) and nt is the number of shared memory processes (OpenMP processes or threads).

4. Numerical Results

In this section, we analyse the parallel TLBO algorithms, presented in Section 3. To perform the tests, we developed the reference algorithm, presented in [2], in C language to implement the parallel algorithms, and used the GCC v.4.8.5 compiler [19]. We chose MPI v2.2 [20] for the high level parallel approach and OpenMP API v3.1 [21] for the shared memory parallel algorithms. The parallel platform used was composed of HP Proliant SL390 G7 nodes, where each node was equipped with two Intel Xeon X5660 processors. Each X5660 included six processing cores at 2.8 GHz, and QDR Infiniband was used as the communication network. The performance was analysed using 30 unconstrained functions, listed and described in Tables 1 and 2.

Table 1. Benchmark functions.

Id.	Name	Dim. (V)	Domain (Min, Max)
F1	Sphere	30	$-100, 100$
F2	SumSquares	30	$-10, 10$
F3	Beale	2	$-4.5, 4.5$
F4	Easom	2	$-100, 100$
F5	Matyas	2	$-10, 10$
F6	Colville	4	$-10, 10$
F7	Trid 6	6	$-V^2, V^2$
F8	Trid 10	10	$-V^2, V^2$
F9	Zakharov	10	$-5, 10$
F10	Schwefel problem 1.2	30	$-100, 100$
F11	Rosenbrock	30	$-30, 30$
F12	Dixon-Price	5	$-10, 10$
F13	Foxholes	2	$-2^{16}, 2^{16}$
F14	Branin	2	$x_1 : -5, 10; x_2 : 0, 15$
F15	Bohachevsky_1	2	$-100, 100$
F16	Booth	2	$-10, 10$
F17	Michalewicz_2	2	$0, \pi$
F18	Michalewicz_5	5	$0, \pi$
F19	Bohachevsky_2	2	$-100, 100$
F20	Bohachevsky_3	2	$-100, 100$
F21	GoldStein-Price	2	$-2, 2$
F22	Perm	4	$-V, V$
F23	Hartman_3	3	$0, 1$
F24	Ackley	30	$-32, 32$
F25	Penalized_2	30	$-50, 50$
F26	Langermann_2	2	$0, 10$
F27	Langermann_5	5	$0, 10$
F28	Langermann_10	10	$0, 10$
F29	Fletcher-Powell_5	5	$x_i, \alpha_i : -\pi, \pi; a_{ij}, b_{ij} : -100, 100$
F30	Fletcher-Powell_10	10	$x_i, \alpha_i : -\pi, \pi; a_{ij}, b_{ij} : -100, 100$

Table 2. Benchmark functions.

Id.	Function
F1	$f = \sum_{i=1}^{V} x_i^2$
F2	$f = \sum_{i=1}^{V} i x_i^2$
F3	$f = (1.5 - x_1 + x_1 x_2)^2 + (2.25 - x_1 + x_1 x_2^2)^2$ $+(2.625 - x_1 + x_1 x_2^3)^2$
F4	$f = -\cos(x_1)\cos(x_2)\exp\left(-(x_1 - \pi)^2 - (x_2 - \pi)^2\right)$
F5	$f = 0.26(x_1^2 + x_2^2) - 0.48 x_1 x_2$
F6	$f = 100(x_1^2 - x_2)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 + 90(x_3^2 - x_4)^2$ $+10.1\left((x_2 - 1)^2 + (x_4 - 1)^2\right) + 19.8(x_2 - 1)(x_4 - 1)$
F7 F8	$f = \sum_{i=1}^{V}(x_i - 1)^2 - \sum_{i=2}^{V} x_i x_{i-1}$
F9	$f = \sum_{i=1}^{V} x_i^2 + \left(\sum_{i=1}^{V} 0.5 i x_i\right)^2 + \left(\sum_{i=1}^{V} 0.5 i x_i\right)^4$
F10	$f = \sum_{i=1}^{V}\left(\sum_{j=1}^{i} x_j\right)^2$
F11	$f = \sum_{i=1}^{V-1}\left(100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right)$
F12	$f = (x_1 - 1)^2 + \sum_{i=2}^{V} i\left(2x_i^2 - x_{i-1}\right)^2$
F13	$f = \left[\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2}(x_i - a_{ij})^6}\right]^{-1}$
F14	$f = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10$
F15	$f = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7$
F16	$f = (x_1 - 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$
F17 F18	$f = -\sum_{i=1}^{V}\sin x_i\left(\sin\left(\frac{i x_i^2}{\pi}\right)\right)^{20}$
F19	$f = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1)\cos(4\pi x_2) + 0.3$
F20	$f = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1 + 4\pi x_2) + 0.3$
F21	$f = \left[1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1 x_2 + 3x_2^2)\right]$ $\left[30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1 x_2 + 27x_2^2)\right]$
F22	$f = \sum_{j=1}^{V}\left[\sum_{i=1}^{i}(i^j + \beta)\left(\left(\frac{x_i}{i}\right)^j - 1\right)\right]^2$
F23	$f = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{3} a_{ij}(x_j - p_{ij})^2\right]$
F24	$f = -20\exp\left(-0.2\sqrt{\frac{1}{V}\sum_{i=1}^{V} x_i^2}\right) - \exp\left(\frac{1}{V}\sum_{i=1}^{V}\cos(2\pi x_i)\right) + 20 + e$

Table 2. *Cont.*

Id.	Function
F25	$f = 0.1\{\sin^2(3\pi x_1) + \sum_{i=1}^{V-1}(x_i-1)^2\left[1+\sin^2(3\pi x_{i+1})\right]$ $+(x_V-1)^2\left[1+\sin^2(2\pi x_V)\right]\} + \sum_{i=1}^{V} u(x_i,5,100,4),$ $u(x_i,a,k,m) =$ $k(x_i-a)^m, x_i > a; 0, -a \le x_i \le a; k(-x_i-a)^m, x_i < -a.$
F26 F27 F28	$f = -\sum_{i=1}^{5} c_i \left[\exp\left(-\frac{1}{\pi}\sum_{j=1}^{V}(x_j-a_{ij})^2\right)\cos\left(\pi\sum_{j=1}^{V}(x_j-a_{ij})^2\right)\right]$
F29	$f = \sum_{i=1}^{V}(C_i-D_i)^2;$ $C_i = \sum_{j=1}^{V}\left(a_{ij}\sin\alpha_j + bij\cos\alpha_j\right),$ $D_i = \sum_{j=1}^{V}\left(a_{ij}\sin x_j + bij\cos x_j\right),$ $a_{i,j}, b_{i,j} = \begin{bmatrix} -79 & 56 & -62 & -9 & 92 \\ 91 & -9 & -18 & -59 & 99 \\ -38 & 8 & -12 & -73 & 40 \\ -78 & -18 & -49 & 65 & 66 \\ -1 & -43 & 93 & -18 & -76 \end{bmatrix},$ $\alpha_j = \begin{bmatrix} -2.791 & 2.5623 & -1.0429 & 0.5097 & -2.8096 \end{bmatrix}.$
F30	$f = \sum_{i=1}^{V}(C_i-D_i)^2;$ $C_i = \sum_{j=1}^{V}\left(a_{ij}\sin\alpha_j + bij\cos\alpha_j\right),$ $D_i = \sum_{j=1}^{V}\left(a_{ij}\sin x_j + bij\cos x_j\right),$ $a_{i,j}, b_{i,j} = \begin{bmatrix} -79 & 56 & -62 & -9 & 92 & 48 & -22 & -34 & -39 & -40 \\ 91 & -9 & -18 & -59 & 99 & -45 & 88 & -14 & -29 & 26 \\ -38 & 8 & -12 & -73 & 40 & 26 & -64 & 29 & -82 & -32 \\ -78 & -18 & -49 & 65 & 66 & -40 & 88 & -95 & -57 & 10 \\ -1 & -43 & 93 & -18 & -76 & -68 & -42 & 22 & 46 & -14 \\ 34 & -96 & 26 & -56 & -36 & -85 & -62 & 13 & 93 & 78 \\ 52 & -46 & -69 & 99 & -47 & -72 & -11 & 55 & -55 & 91 \\ 81 & 47 & 35 & 55 & 67 & -13 & 33 & 14 & 83 & -42 \\ 5 & -43 & -45 & 46 & 56 & -94 & -62 & 52 & 66 & 55 \\ -50 & 66 & -47 & -75 & 89 & -16 & 82 & 6 & -85 & -62 \end{bmatrix},$ $\alpha_j = \begin{bmatrix} -2.791 & 2.5623 & -1.0429 & 0.5097 & -2.8096 & \dots \\ \dots & 1.1883 & 2.0771 & -2.9926 & 0.0715 & 0.4142 \end{bmatrix}.$

We will now analyse parallel behaviour of the parallel algorithm SPG_TLBO, described in Algorithms 4–6, i.e., the shared memory parallel algorithm that stores the whole population in shared (or global) memory. Table 3 shows the parallel efficiencies for all functions of the benchmark test, using a number of threads (NoT) between 2 to 10. In this table, we can see that good efficiencies are obtained for almost all functions using up to 6 threads. However, in very low computational cost functions, efficiency decreases rather considerably when increasing the number of threads. In such

cases, to be able to increase the number of processes efficiently, the heterogeneous memory parallel TLBO algorithm should be used.

Table 3. Efficiencies for SPG_TLBO, Runs = 30, Population size = 240, Iterations = 1000.

Function	NoT	Efficiency	Function	NoT	Efficiency	Function	NoT	Efficiency
F1	2	80%	F11	2	81%	F21	2	71%
	4	64%		4	66%		4	50%
	6	58%		6	59%		6	36%
	8	52%		8	54%		8	26%
	10	47%		10	49%		10	22%
F2	2	82%	F12	2	63%	F22	2	93%
	4	69%		4	37%		4	85%
	6	60%		6	26%		6	73%
	8	55%		8	19%		8	65%
	10	50%		10	15%		10	59%
F3	2	71%	F13	2	82%	F23	2	70%
	4	45%		4	79%		4	57%
	6	31%		6	74%		6	46%
	8	22%		8	70%		8	42%
	10	17%		10	67%		10	37%
F4	2	70%	F14	2	66%	F24	2	66%
	4	56%		4	45%		4	57%
	6	46%		6	30%		6	54%
	8	37%		8	21%		8	50%
	10	32%		10	16%		10	50%
F5	2	76%	F15	2	65%	F25	2	67%
	4	64%		4	52%		4	62%
	6	54%		6	41%		6	59%
	8	42%		8	33%		8	49%
	10	38%		10	27%		10	49%
F6	2	63%	F16	2	65%	F26	2	77%
	4	36%		4	34%		4	59%
	6	21%		6	21%		6	45%
	8	16%		8	14%		8	36%
	10	12%		10	11%		10	31%
F7	2	68%	F17	2	77%	F27	2	76%
	4	58%		4	59%		4	67%
	6	52%		6	44%		6	62%
	8	49%		8	33%		8	55%
	10	46%		10	27%		10	53%
F8	2	64%	F18	2	74%	F28	2	65%
	4	55%		4	64%		4	50%
	6	48%		6	58%		6	56%
	8	45%		8	55%		8	60%
	10	42%		10	53%		10	46%
F9	2	73%	F19	2	63%	F29	2	96%
	4	59%		4	50%		4	77%
	6	42%		6	39%		6	64%
	8	36%		8	31%		8	62%
	10	31%		10	27%		10	49%
F10	2	90%	F20	2	67%	F30	2	93%
	4	81%		4	51%		4	88%
	6	73%		6	41%		6	82%
	8	70%		8	33%		8	81%
	10	65%		10	28%		10	81%

Tables 4 and 5 show the parallel efficiencies for the heterogeneous memory parallel TLBO algorithm using SPG_TLBO, setting the number of total processes (NoTP) to 4 and 10, when the number of (MPI) processes (NoP) is equal to 1 the SPG_TLBO algorithm is used. We compare the SPG_TLBO parallel algorithm with respect the hybrid MPI/OpenMP algorithm using the same number of total processes (NoTP). Since the MPI algorithm is independent of the OpenMP algorithm, the same behaviour is obtained when the SPP_TLBO algorithm is used instead of the SPG_TLBO.

Table 4. Efficiencies for hybrid MPI/OpenMP parallel algorithm, Runs = 30, Population size = 240, Iterations = 1000.

Func.	NoP	NoT	NoTP	Eff.	NoP	NoT	NoTP	Eff.
F1	1	4	4	64%	1	10	10	47%
	2	2		73%	5	2		73%
F2	1	4	4	69%	1	10	10	50%
	2	2		77%	5	2		74%
F3	1	4	4	45%	1	10	10	17%
	2	2		64%	5	2		64%
F4	1	4	4	56%	1	10	10	32%
	2	2		64%	5	2		61%
F5	1	4	4	53%	1	10	10	38%
	2	2		64%	5	2		58%
F6	1	4	4	36%	1	10	10	12%
	2	2		66%	5	2		62%
F7	1	4	4	58%	1	10	10	46%
	2	2		66%	5	2		61%
F8	1	4	4	55%	1	10	10	42%
	2	2		60%	5	2		59%
F9	1	4	4	59%	1	10	10	31%
	2	2		62%	5	2		64%
F10	1	4	4	51%	1	10	10	65%
	2	2		86%	5	2		80%
F11	1	4	4	66%	1	10	10	49%
	2	2		77%	5	2		73%
F12	1	4	4	37%	1	10	10	15%
	2	2		55%	5	2		59%
F13	1	4	4	79%	1	10	10	67%
	2	2		85%	5	2		91%
F14	1	4	4	45%	1	10	10	16%
	2	2		60%	5	2		63%
F15	1	4	4	52%	1	10	10	27%
	2	2		62%	5	2		62%

As can be seen, using the hybrid MPI/OpenMP algorithm can significantly increase scalability of the parallel algorithm. Table 6 shows the efficiencies for highest computational cost functions, increasing the total number of processes (NoTP) to 30 and the number of iterations to 10,000, and in which the good behaviour of the efficiency can be verified.

Table 5. Efficiencies for hybrid MPI/OpenMP parallel algorithm, Runs = 30, Population size = 240, Iterations = 1000.

Func.	NoP	NoT	NoTP	Eff.	NoP	NoT	NoTP	Eff.
F16	1	4	4	64%	1	10	10	47%
	2	2		73%	5	2		73%
F17	1	4	4	64%	1	10	10	47%
	2	2		73%	5	2		73%
F18	1	4	4	64%	1	10	10	47%
	2	2		73%	5	2		73%
F19	1	4	4	64%	1	10	10	47%
	2	2		73%	5	2		73%
F20	1	4	4	64%	1	10	10	47%
	2	2		73%	5	2		73%
F21	1	4	4	64%	1	10	10	47%
	2	2		73%	5	2		73%
F22	1	4	4	64%	1	10	10	47%
	2	2		73%	5	2		73%
F23	1	4	4	64%	1	10	10	47%
	2	2		73%	5	2		73%
F24	1	4	4	64%	1	10	10	47%
	2	2		63%	5	2		73%
F25	1	4	4	64%	1	10	10	47%
	2	2		73%	5	2		73%
F26	1	4	4	64%	1	10	10	47%
	2	2		73%	5	2		73%
F27	1	4	4	64%	1	10	10	47%
	2	2		73%	5	2		73%
F28	1	4	4	64%	1	10	10	47%
	2	2		73%	5	2		73%
F29	1	4	4	64%	1	10	10	47%
	2	2		73%	5	2		73%
F30	1	4	4	64%	1	10	10	47%
	2	2		73%	5	2		73%

Table 6. Efficiencies for hybrid MPI/OpenMP parallel algorithm, Runs = 30, Population size = 240, Iterations = 10,000, NoP = 15, NoT = 2, NoTP = 30.

Function	Efficiency	Function	Efficiency	Function	Efficiency
F4	64%	F15	64%	F25	90%
F5	71%	F18	71%	F26	95%
F7	65%	F19	66%	F27	69%
F8	66%	F22	93%	F28	70%
F10	85%	F23	68%	F29	86%
F13	92%	F24	63%	F30	93%

Table 7 shows the parallel efficiencies for the SPP_TLBO algorithm using the same sequential reference algorithm as the one used in Table 7, i.e., the sequential TLBO algorithm. Worthy of note, the parallel algorithm SPP_TLBO does not emulate the sequential algorithm TLBO literally. The use of subpopulations in the SPP_TLBO algorithm causes modifications in some procedures, such as the calculation of the mean of the variables; it also reduces the working population in some procedures, such as the detection of duplicates. This means that on the one hand, efficiency results generally improve with respect to the SPG_TLBO algorithm, and on the other, in some cases, the efficiency exceeds the theoretical upper limit when comparing exactly the same algorithms. In particular the

duplicate removal procedure for very low cost computational cost functions becomes a very important aspect in the overall cost of the algorithm.

Table 7. Efficiencies for SPP_TLBO, Runs = 30, Population size = 120, Iterations = 1000.

Function	NoT	Efficiency	Function	NoT	Efficiency	Function	NoT	Efficiency
F1	2	97%	F11	2	96%	F21	2	137%
	4	87%		4	87%		4	122%
	6	80%		6	81%		6	107%
	8	73%		8	76%		8	83%
	10	69%		10	70%		10	65%
F2	2	96%	F12	2	107%	F22	2	97%
	4	86%		4	82%		4	91%
	6	81%		6	69%		6	86%
	8	75%		8	56%		8	84%
	10	71%		10	44%		10	82%
F3	2	105%	F13	2	91%	F23	2	121%
	4	89%		4	86%		4	121%
	6	78%		6	85%		6	111%
	8	65%		8	84%		8	110%
	10	52%		10	84%		10	94%
F4	2	135%	F14	2	122%	F24	2	46%
	4	143%		4	106%		4	50%
	6	139%		6	96%		6	73%
	8	118%		8	76%		8	80%
	10	102%		10	61%		10	79%
F5	2	113%	F15	2	158%	F25	2	94%
	4	80%		4	175%		4	88%
	6	63%		6	166%		6	83%
	8	46%		8	135%		8	80%
	10	34%		10	110%		10	77%
F6	2	109%	F16	2	112%	F26	2	106%
	4	81%		4	78%		4	102%
	6	67%		6	64%		6	100%
	8	54%		8	47%		8	94%
	10	41%		10	35%		10	88%
F7	2	136%	F17	2	115%	F27	2	121%
	4	130%		4	114%		4	117%
	6	132%		6	107%		6	111%
	8	107%		8	100%		8	107%
	10	94%		10	90%		10	98%
F8	2	105%	F18	2	115%	F28	2	107%
	4	86%		4	111%		4	105%
	6	78%		6	113%		6	102%
	8	66%		8	109%		8	98%
	10	56%		10	106%		10	94%
F9	2	100%	F19	2	114%	F29	2	81%
	4	81%		4	113%		4	80%
	6	72%		6	109%		6	74%
	8	67%		8	103%		8	66%
	10	60%		10	95%		10	70%
F10	2	96%	F20	2	121%	F30	2	97%
	4	88%		4	115%		4	92%
	6	84%		6	108%		6	84%
	8	81%		8	104%		8	90%
	10	78%		10	98%		10	86%

As shown in Tables 3–5 and 7, the parallel methods proposed obtain good efficiencies. However in [14], the authors' parallel proposal for the particular problem under study, also achieves very good efficiencies, the cost function having a high computational cost. In Table 8, we compare the method proposed in [14] to both proposed methods, SPG_TLBO and SPP_TLBO, for the first function

of the benchmark test (provided by the reference software in https://gitlab.hpca.ual.es/ncc911/ParallelTLBO), i.e., the Sphere function, using between 2 to 10 threads (NoT), i.e., OpenMP processes. Results presented in Table 8 were obtained by running the reference code on the same parallel platform where the results for the SPG_TLBO and SPP_TLBO algorithms have been obtained. As shown, the efficiencies for both proposed algorithms, SPG_TLBO and SPP_TLBO, improve those obtained by the reference algorithm, especially by increasing the number of threads used. Worthy of note the TLBO parallel proposal presented in [13] obtains efficiencies of only between 20% and 30% for 16 and 32 processes respectively, and other parallel proposals applied to the state-of-the-art algorithms DPGA and ACO, obtain worse efficiency results and serious scalability problems.

Table 8. Comparison of SPG_TLBO and SPP_TLBO respect to algorithm presented in [14].

Iterations	Pop. Size	NoT	Alg. Ref	SPG_TLBO	SPP_TLBO
1000	60	2	61%	77%	89%
		4	33%	60%	73%
		6	22%	45%	61%
		8	16%	35%	47%
		10	12%	28%	38%
	120	2	65%	76%	97%
		4	38%	64%	87%
		6	26%	54%	80%
		8	19%	43%	73%
		10	15%	36%	64%
	240	2	61%	80%	100%
		4	34%	64%	91%
		6	23%	58%	87%
		8	18%	52%	80%
		10	14%	47%	75%
5000	60	2	61%	74%	89%
		4	35%	56%	68%
		6	22%	42%	56%
		8	17%	33%	44%
		10	13%	26%	36%
	120	2	64%	77%	96%
		4	37%	63%	78%
		6	26%	52%	68%
		8	19%	42%	58%
		10	15%	34%	51%
	240	2	61%	79%	107%
		4	35%	64%	97%
		6	24%	57%	88%
		8	19%	52%	80%
		10	15%	47%	74%
10,000	60	2	62%	74%	89%
		4	35%	56%	67%
		6	22%	42%	55%
		8	17%	33%	44%
		10	13%	26%	36%
	120	2	64%	77%	96%
		4	38%	64%	82%
		6	26%	52%	67%
		8	19%	42%	58%
		10	15%	35%	51%
	240	2	61%	80%	111%
		4	34%	64%	101%
		6	23%	58%	88%
		8	18%	52%	80%
		10	14%	47%	75%

Finally, the effectiveness of the optimization, especially of the SPP_TLBO algorithm, should be checked, as it modifies the procedure carried out in the sequential TLBO algorithm, while the

SPG_TLBO algorithm performs a processing that is analogous to the sequential processing. Table 9 show the number of iterations (N. It.) needed to achieve an optimal value with an error of less than $1e-3$. This table shows the data of the original sequentially executed TLBO algorithm and the data of the parallel SPP_TLBO algorithms. Please note that the number of iterations shown in Table 9 is the average of the functions iterations needed to achieve an optimal value with an error of less than $1e-3$, this average has been computed over the 30 values obtained from the 30 independent runs performed, both for the sequential and parallel algorithms. On the other hand, both the subpopulation size and the population size for the sequential algorithm are equal to 120. Whorty of note, the number of iterations when using the SPG_TLBO parallel algorithm is similar to the sequential reference algorithm, due to the sequential procedure has not been modified. While the number of iterations, shown in Table 9, when using the SPP_TLBO parallel algorithm shows that our parallel proposal outperforms the sequential TLBO algorithm, i.e., convergence is accelerated. Therefore, the strategy of using subpopulations connected by the best global individual, used in the SPP_TLBO algorithm, offers improvements both at the computational level and regarding convergence speed. Table 9 does not include those functions of faster convergence.

Table 9. Average number of function iterations for SPP_TLBO, Runs = 30, Subpopulation size = 120.

Function	NoT	N. It.	Function	NoT	N. It.	Function	NoT	N. It.
F1	Seq.	432	F10	Seq.	2001	F22	Seq.	947
	2	474		2	2333		2	538
	4	426		4	1992		4	368
	6	398		6	2289		6	219
	8	388		8	1770		8	219
	10	387		10	2096		10	427
F2	Seq.	507	F11	Seq.	14,059	F24	Seq.	300
	2	443		2	10,943		2	269
	4	443		4	12,685		4	281
	6	474		6	11,232		6	262
	8	455		8	13,834		8	247
	10	447		10	14,259		10	255
F4	Seq.	32	F12	Seq.	54	F25	Seq.	427
	2	25		2	47		2	333
	4	39		4	42		4	347
	6	33		6	32		6	300
	8	33		8	44		8	324
	10	26		10	33		10	279
F6	Seq.	285	F13	Seq.	277	F26	Seq.	35
	2	269		2	268		2	10
	4	117		4	423		4	20
	6	106		6	398		6	18
	8	101		8	285		8	8
	10	96		10	285		10	6
F7	Seq.	41	F15	Seq.	22	F27	Seq.	62
	2	38		2	17		2	69
	4	34		4	20		4	45
	6	37		6	18		6	40
	8	35		8	18		8	41
	10	33		10	15		10	35
F8	Seq.	282	F18	Seq.	54	F29	Seq.	67
	2	262		2	132		2	58
	4	206		4	74		4	73
	6	307		6	66		6	45
	8	146		8	91		8	41
	10	176		10	75		10	3
F9	Seq.	209	F19	Seq.	16	F30	Seq.	657
	2	176		2	19		2	216
	4	164		4	18		4	410
	6	173		6	16		6	176
	8	158		8	18		8	221
	10	148		10	16		10	155

5. Conclusions

The TLBO heuristic optimization algorithm is an effective optimization algorithm that though recent, has been tested and compared. In this work, we presented efficient parallel algorithms for heterogeneous parallel platforms. We proposed a hybrid MPI/OpenMP algorithm, exploiting inherent parallelism at different levels. Moreover we proposed two different algorithms for shared memory architectures, using OpenMP, called SPG_TLBO and SPP_TLBO. The first is an efficient parallel implementation of the TLBO sequential algorithm without any changes to the sequential procedure. In the second, SPP_TLBO, we proposed a different strategy that improves both computational performance and optimization behaviour. Significantly, the parallel proposals achieved good parallel performance regardless of the intrinsic characteristics of the functions to be optimized, in particular with regard to the computational cost of the function to be optimized. On the other hand, the high level parallel proposal included an intrinsic load balancing mechanism allowing the use of non-dedicated computing platforms.

Author Contributions: H.M., A.G.-M., J.-L.S.-R. and R.V.R. conceived the parallel algorithms; R.V.R. conceived the sequential algorithm; A.G.-M. and H.M. designed parallel algorithms; H.M. and A.G.-M. codified the parallel algorithms; A.G.-M. and H.R. performed numerical experiments; H.M. and A.J.-M. analyzed the data; H.M. and J.-L.S.-R. wrote the paper.

Funding: This research was supported by the Spanish Ministry of Economy and Competitiveness under Grants TIN2015-66972-C5-4-R and TIN2017-89266-R, co-financed by FEDER funds. (MINECO/FEDER/UE).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Lin, M.H.; Tsai, J.F.; Yu, C.S. A Review of Deterministic Optimization Methods in Engineering and Management. *Math. Probl. Eng.* **2012**, *2012*, 756023. [CrossRef]
2. Rao, R.V.; Patel, V. Comparative performance of an elitist teaching-learning-based optimization algorithm for solving unconstrained optimization problems. *Int. J. Ind. Eng. Comput.* **2013**, *4*, 29–50. [CrossRef]
3. Singh, M.; Panigrahi, B.; Abhyankar, A. Optimal coordination of directional over-current relays using Teaching Learning-Based Optimization (TLBO) algorithm. *Int. J. Electr. Power Energy Syst.* **2013**, *50*, 33–41. [CrossRef]
4. Niknam, T.; Azizipanah-Abarghooee, R.; Narimani, M.R. A new multi objective optimization approach based on TLBO for location of automatic voltage regulators in distribution systems. *Eng. Appl. Artif. Intell.* **2012**, *25*, 1577–1588. [CrossRef]
5. Li, D.; Zhang, C.; Shao, X.; Lin, W. A multi-objective TLBO algorithm for balancing two-sided assembly line with multiple constraints. *J. Intell. Manuf.* **2016**, *27*, 725–739. [CrossRef]
6. Arya, L.; Koshti, A. Anticipatory load shedding for line overload alleviation using Teaching learning based optimization (TLBO). *Int. J. Electr. Power Energy Syst.* **2014**, *63*, 862–877. [CrossRef]
7. Mohanty, B. TLBO optimized sliding mode controller for multi-area multi-source nonlinear interconnected AGC system. *Int. J. Electr. Power Energy Syst.* **2015**, *73*, 872–881. [CrossRef]
8. Yan, J.; Li, K.; Bai, E.; Yang, Z.; Foley, A. Time series wind power forecasting based on variant Gaussian Process and TLBO. *Neurocomputing* **2016**, *189*, 135–144. [CrossRef]
9. Baghban, A.; Kardani, M.N.; Mohammadi, A.H. Improved estimation of Cetane number of fatty acid methyl esters (FAMEs) based biodiesels using TLBO-NN and PSO-NN models. *Fuel* **2018**, *232*, 620–631. [CrossRef]
10. Javaid, N.; Ahmed, A.; Iqbal, S.; Ashraf, M. Day Ahead Real Time Pricing and Critical Peak Pricing Based Power Scheduling for Smart Homes with Different Duty Cycles. *Energies* **2018**, *11*, 1464. [CrossRef]
11. Zakeri, A.S.; Askarian Abyaneh, H. Transmission Expansion Planning Using TLBO Algorithm in the Presence of Demand Response Resources. *Energies* **2017**, *10*, 1376. [CrossRef]
12. Rao, R.V.; Savsani, V.; Vakharia, D. Teaching-learning-based optimization: A novel method for constrained mechanical design optimization problems. *Comput.-Aided Des.* **2011**, *43*, 303–315. [CrossRef]
13. Umbarkar, A.J.; Rothe, N.M.; Sathe, A. OpenMP Teaching-Learning Based Optimization Algorithm over Multi-Core System. *Int. J. Intell. Syst. Appl.* **2015**, *7*, 19–34. [CrossRef]

14. Cruz, N.C.; Redondo, J.L.; Álvarez, J.D.; Berenguel, M.; Ortigosa, P.M. A parallel Teaching–Learning-Based Optimization procedure for automatic heliostat aiming. *J. Supercomput.* **2017**, *73*, 591–606.
15. Umbarkar, A.J.; Joshi, M.S.; Sheth, P.D. OpenMP Dual Population Genetic Algorithm for Solving Constrained Optimization Problems. *Int. J. Inf. Eng. Electron. Bus.* **2015**, *1*, 59–65. [CrossRef]
16. Delisle, P.; Krajecki, M.; Gravel, M.; Gagné, C. Parallel implementation of an ant colony optimization metaheuristic with OpenMP. In Proceedings of the 3rd European Workshop on OpenMP, Barcelona, Spain, 8–9 September 2001; Springer: Berlin/Heidelberg, Germany, 2001.
17. Lee, H.; Kim, K.; Kwon, Y.; Hong, E. Real-Time Particle Swarm Optimization on FPGA for the Optimal Message-Chain Structure. *Electronics* **2018**, *7*, 274. [CrossRef]
18. Jimeno-Morenilla, A.; Sánchez-Romero, J.L.; Migallón, H.; Mora-Mora, H. Jaya optimization algorithm with GPU acceleration. *J. Supercomput.* **2018**. [CrossRef]
19. Free Software Foundation, Inc. GCC, the GNU Compiler Collection. Available online: https://www.gnu.org/software/gcc/index.html (accessed on 2 November 2016).
20. MPI Forum. MPI: A Message-Passing Interface Standard. Version 2.2. 2009. Available online: http://www.mpi-forum.org (accessed on 15 December 2016).
21. OpenMP Architecture Review Board. OpenMP Application Program Interface, Version 3.1. 2011. Available online: http://www.openmp.org (accessed on 2 November 2016).