Applied Soft Computing 12 (2012) 3552–3579

Contents lists available at SciVerse ScienceDirect

Applied Soft Computing

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

Parallel cooperative micro-particle swarm optimization: A master–slave model

Konstantinos E. Parsopoulos*

Department of Computer Science, University of Ioannina, P.O. Box 1186, GR-45110 Ioannina, Greece

ARTICLE INFO

Article history:
Received 29 December 2011
Received in revised form 11 May 2012
Accepted 2 July 2012
Available online 31 July 2012

Keywords:
Particle swarm optimization
Parallel algorithms
Cooperative algorithms
Master–slave model
Micro-evolutionary algorithms

ABSTRACT

A parallel master–slave model of the recently proposed cooperative micro-particle swarm optimization approach is introduced. The algorithm is based on the decomposition of the original search space in subspaces of smaller dimension. Each subspace is probed by a subswarm of small size that identifies suboptimal partial solution components. A context vector that serves as repository for the best attained partial solutions of all subswarms is used for the evaluation of the particles. The required modifications to fit the original algorithm within a parallel computation framework are discussed along with their impact on performance. Also, both linear and random allocation of direction components to subswarms are considered to render the algorithm capable of capturing possible correlations among decision variables. The proposed approach is evaluated on two types of computer systems, namely an academic cluster and a desktop multicore system, using a popular test suite. Statistical analysis of the obtained results reveals that, besides the expected run-time superiority of the parallel model, significant improvements in solution quality can also be achieved. Different factors that may affect performance are pointed out, offering intuition on the expected behavior of the parallel model.

1. Introduction

Particle swarm optimization (PSO) has been shown to be an efficient optimization algorithm in a plethora of global optimization tasks. Its efficiency and easy implementation has rendered PSO a very popular approach. Today, its applicability spans a large number of scientific and technological fields, including power systems [1], electromagnetics [2], circuit design [3], antenna design [4], machine learning [5], scheduling [6], bioinformatics and medical informatics [7,8], and astrophysics [9]. Numerous applications of PSO are reported in relevant surveys [10–12] and its most crucial features are analyzed in specialized books [13–15].

However, as it holds true for most metaheuristic optimization algorithms, the efficiency of PSO deteriorates proportionally to the dimensionality of the problem. This property is usually referred to as the *curse of dimensionality*, adopting the corresponding nomenclature from dynamic programming [16]. Unfortunately, modern applications often involve optimization problems of high dimensionality and complexity. In such problems, standard variants of metaheuristics usually exhibit inferior performance.

This deficiency is often attributed to the algorithms' inability to tackle complex, high-dimensional, nonlinear objective functions and vast search spaces by using a single search module, i.e., a single population. The problem has been addressed by exploiting the inherent parallelization properties of most metaheuristics. New variants that employ multiple search modules have been developed. Typically, the search modules are designed to concurrently acquire and share information in a cooperative manner. These developments gave rise to the class of *cooperative algorithms* [17].

Up-to-date, various established optimization approaches have been extended within a cooperative framework, including simulated annealing [18], tabu search [19], genetic algorithms [20], ant colony optimization [21], evolution strategies [22] and PSO [23]. A comprehensive survey of the most promising PSO-based cooperative models can be found in [24]. Experimental evidence suggests that, in many cases, the cooperative models can achieve superior performance than the corresponding original algorithms under the same computational budget limitations.

This property has triggered research towards the development of cooperative variants that can achieve similar performance levels with the original algorithms, although requiring only a fraction of the available computational resources. Such approaches could extend the applicability of evolutionary algorithms in low-end computer systems where demanding applications are prohibitive. The first step towards this direction was the development of rudimentary instances of the standard evolutionary algorithms, called *micro-evolutionary algorithms* (micro-EAs), which are characterized by small population sizes and simple operators.

Although micro-EAs were primarily considered for educational purposes, recent studies revealed their potential as promising light-weight optimizers even for demanding applications such as image processing [25]. In this context, a micro-PSO approach was

* Tel.: +30 2651008839; fax: +30 2651008890.
E-mail address: kostasp@cs.uoi.gr

1568-4946/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.asoc.2012.07.013

introduced [26] and shown to be a viable alternative for tackling high-dimensional problems. The most common deficiency of micro-EAs was the rapid loss of diversity due the small population sizes, which often resulted in search stagnation. This problem was addressed by incorporating proper diversity-preserving techniques to enhance the exploration capability of the algorithms.

Recently, the efficiency of cooperative algorithms on high-dimensional problems was combined with the flexibility and low computational requirements of micro-PSO in the *cooperative micro-particle swarm optimization* (COMPSO) algorithm [27]. In COMPSO, the search space is linearly decomposed in subspaces of lower dimension. Then, each subspace is assigned to a subswarm and concurrently probed with the rest in a cooperative manner.

The particles of each subswarm carry only partial solution information. Hence, the construction of a complete solution vector requires cooperation and information sharing among the subswarms. This is achieved by using a common context vector, where all subswarms deposit their best findings. The context vector is used for the evaluation of the particles of all subswarms. Thus, the search of each subswarm is indirectly influenced by the rest. Preliminary experiments indicated that COMPSO could significantly outperform standard PSO in widely used benchmark problems of dimension up to 1200 [27]. The same cooperative model was also used with the differential evolution algorithm, exhibiting promising results [28].

Besides the increased efficiency, cooperative algorithms possess an additional desirable feature: they can easily fit a *high-performance computing* (HPC) environment. HPC systems have offered the ground for the study of previously intractable complex problems by tremendously increasing the computational power and storage resources. Also, they offered the technical background and experience for the development of more efficient processors that incorporate inherent parallelization properties. Such processors are available in modern multicore personal computers (PCs), which can efficiently tackle demanding computational tasks, providing small-scale, shared-memory, parallel-computing solutions to the user. The parallelization capabilities can be exploited by using widely used software such as the *message passing interface*[1] (MPI) standard and the *parallel virtual machine*[2] (PVM).

The present paper introduces a parallel master–slave model of COMPSO (henceforth denoted as PCOMPSO). To the best of the author's knowledge, this is the first parallel implementation of a cooperative approach based on micro-PSO. For this reason, the simple yet efficient master–slave parallelization model was used. The algorithm was implemented following the MPI standard and it was assessed on a widely used test suite. Since parallelization is primarily recommended for the optimization of time-consuming objective functions, random time-delay was added to each function evaluation to simulate computationally demanding objective functions.

In addition to the linear decomposition scheme that is used in the original COMPSO [27], a random decomposition scheme was considered in PCOMPSO to tackle possible correlations among the problem's decision variables. The proposed approach was implemented and validated on two different computer systems. The first one was the Saw facility of the *shared hierarchical academic research computing network*[3] (SHARCNET), which is an academic cluster for parallel computations. The second one was a shared-memory multicore PC. The results were statistically analyzed with respect to solution quality and time-performance. The present study primarily aimed at:

(a) revealing possible superiority of PCOMPSO against COMPSO,
(b) investigating the impact of modifications in the original COMPSO imposed by the parallelization,
(c) assessing the random decomposition scheme against the linear one, and
(d) gaining intuition regarding PCOMPSO's performance on different parallel computation environments.

The rest of the paper is organized as follows: Section 2 provides the necessary background information, while the proposed master–slave model of PCOMPSO is analyzed in Section 3. The experimentation environments and parameter settings are reported in Section 4, followed by the experimental results and discussion in Section 5. Finally, the paper concludes in Section 6.

2. Background information

In the following paragraphs, the necessary background information is briefly described. This includes descriptions of PSO, micro-PSO, their cooperative extensions as well as a presentation of the COMPSO algorithm.

2.1. Particle swarm optimization

PSO is a stochastic population-based optimization algorithm. It was introduced in 1995 as a new approach for solving continuous optimization problems, inspired by the collective behavior and emergent phenomena in decentralized systems [29]. Today, PSO is an exceptional representative of *swarm intelligence* algorithms [30].

Putting it formally, let the minimization problem:

$$\min_{x\in X} f(x),$$

where $f : X \subset \mathbb{R}^n \to \mathbb{R}$ is an n-dimensional objective function. PSO probes the search space X with a set of search agents. This set is called a *swarm* while the search agents are called the *particles*. Thus, if the swarm consists of N particles, it can be defined as:

$$\mathcal{S} = \{x_1, x_2, \ldots, x_N\},$$

where the ith particle is an n-dimensional vector:

$$x_i = (x_{i1}, x_{i2}, \ldots, x_{in})^\top \in X, \qquad i \in I = \{1, 2, \ldots, N\}.$$

The particles move in the search space by assuming an adaptable position shift, called *velocity*:

$$v_i = (v_{i1}, v_{i2}, \ldots, v_{in})^\top, \quad i \in I,$$

while retaining in memory the *best position* they have ever attained in X:

$$p_i = (p_{i1}, p_{i2}, \ldots, p_{in})^\top \in X, \qquad i \in I.$$

Moreover, each particle assumes a *neighborhood* of other particles with which it shares its findings. The best position achieved by all the members of a neighborhood is used for their velocity update. Neighborhoods are defined according to *neighborhood topologies*, which define the communication channels among the particles [31].

The *ring* is probably the most popular neighborhood topology. According to it, if r is the neighborhood *radius*, then the ith particle shares information with the particles with indices belonging in the set:

$$NB_{i,r} = \{i - r, i - r + 1, \ldots, i, \ldots, i + r - 1, i + r\},$$

[1] http://www.mcs.anl.gov/research/projects/mpi/
[2] http://www.csm.ornl.gov/pvm/
[3] http://www.sharcnet.ca/

where the index 1 is considered as the immediate neighbor after N. Let g_i be the index of the particle with the best position among all in $NB_{i,r}$, i.e.:

$$g_i = arg \min_{k \in NB_{i,r}} \{f(p_k)\},$$

and let t be the iteration counter. Then, the swarm is manipulated by the following equations [32]:

$$v_{ij}(t+1) = \chi[v_{ij}(t) + c_1\mathcal{R}_1(p_{ij}(t) - x_{ij}(t)) + c_2\mathcal{R}_2(p_{g_i,j}(t) - x_{ij}(t))], \tag{1}$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1), \tag{2}$$

where $i \in I$, $j = 1, 2, \ldots, n$; χ is a parameter called the *constriction coefficient*; c_1 and c_2 are the *cognitive* and *social* parameters, respectively; and $\mathcal{R}_1$, $\mathcal{R}_2$, are random variables uniformly distributed in [0,1]. Notice that a different value of $\mathcal{R}_1$ and $\mathcal{R}_2$ is sampled for each i and j in Eq. (1). The best positions are updated at each iteration as follows:

$$p_i(t+1) = \begin{cases} x_i(t+1), & \text{if } f(x_i(t+1)) < f(p_i(t)), \\ p_i(t), & \text{otherwise.} \end{cases} \tag{3}$$

The constriction coefficient χ in Eq. (1) is a mechanism for controlling the magnitude of velocities. It was added to the original PSO model as a means of addressing the problem of *swarm explosion*, while its properties were studied in the stability analysis of PSO [32,33]. Stemming from this study, the parameter values:

$$\chi = 0.729, \quad c_1 = c_2 = 2.05, \tag{4}$$

are considered as the default parameter setting in many PSO approaches.

2.2. Micro-particle swarm optimization

Micro-PSO has the same structure with standard PSO. The only difference lies in the considered swarm size, N, which shall be small in the case of micro–PSO. Typically, it is required that $N \leq 10$, with $N = 5$ being a popular choice. An identified drawback of selecting small swarm size is the premature convergence of the particles to the best detected solutions within a few iterations of the algorithm. As expected, this effect becomes more severe in highly nonlinear problems as the dimension, n, increases.

Attempting to quantify the difficulty of a problem, the following ratio can be defined:

$$D = \frac{n}{N}. \tag{5}$$

Empirical evidence for EAs suggest that, in most cases, the problem becomes harder as D increases. Based on this finding, it is expected that micro-PSO may raise performance issues in high-dimensional problems.

In general, the weakness of search stagnation and premature convergence of micro-EAs has been addressed by restarting the population as soon as there is no further progress in search. Also, specialized techniques were introduced to alleviate the repetitive convergence of the algorithm to previous solutions after restarting. For instance, in [26] μPSO was equipped with a repelling scheme that keeps the swarm away from any point belonging in a black-list of detected solutions. This technique has many in common with tabu search approaches as well as with the repulsion technique introduced in [34]. Nevertheless, the small amount of reported results for micro-PSO, mostly for test problems of dimension $n \leq 500$, has proved to be inadequate to establish it as a promising alternative for solving difficult high-dimensional problems.

2.3. Cooperative particle swarm optimization

Cooperative PSO (CPSO) variants [35,36] have been proposed as efficient approaches for solving high-dimensional problems by decomposing the original n-dimensional search space X into low-dimensional subspaces:

$$X = X_1 \times X_2 \times \cdots \times X_K,$$

each one probed by a different subswarm. More specifically, let n_1, n_2, …, n_K, be K positive integers indicating the dimensions of the subspaces, i.e.:

$$n = \sum_{k=1}^{K} n_k, \qquad n_k \geq 1, \qquad k = 1, 2, \ldots, K,$$

where n is the dimension of the original problem. Then, instead of using a single n-dimensional swarm as in standard PSO, CPSO employs K subswarms, $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_K$, of sizes N_1, N_2, …, N_K, respectively. Each subswarm $\mathcal{S}_k$ undertakes the search of the corresponding n_k-dimensional subspace X_k. For proper choices of n_k and N_k, the corresponding ratio $D_k = n_k/N_k$ as defined in Eq. (5) can become significantly smaller than the original one, implying an easier optimization task in the corresponding n_k-dimensional subspace.

The update of each subswarm in CPSO is identical to the update of standard PSO described in Section 2.1. However, a significant issue arises with the evaluation of the particles. More specifically, since each subswarm operates in a subspace of smaller dimension than the original search space, its particles carry only partial solution information. Thus, they cannot be directly evaluated with the objective function due to the missing components.

This problem is addressed by combining the best information of each subswarm in a shared buffer vector, also called *context vector* [36]. This vector is used as a host of the best findings of all subswarms, providing to each particle the missing information required for its evaluation. This kind of indirect cooperation among the subswarms offers the opportunity to eventually move closer to the best positions of the original search space by combining information gathered from its low-dimensional subspaces.

More specifically, let $\mathcal{B}$ be the n-dimensional buffer vector where each subswarm deposits its contribution. Then, if:

$$b^{[k]} = (b_1^{[k]}, b_2^{[k]}, \ldots, b_{n_k}^{[k]})^\top,$$

is a contributed n_k-dimensional vector by the kth subswarm $\mathcal{S}_k$, the buffer vector is defined as:

$$\mathcal{B} = (\underbrace{b_1^{[1]}, \ldots, b_{n_1}^{[1]}}_{\text{contribution of } \mathcal{S}_1}, \underbrace{b_1^{[2]}, \ldots, b_{n_2}^{[2]}}_{\text{contribution of } \mathcal{S}_2}, \ldots, \underbrace{b_1^{[K]}, \ldots, b_{n_K}^{[K]}}_{\text{contribution of } \mathcal{S}_K})^\top.$$

The ith particle of the jth subswarm, denoted as:

$$x_i^{[j]} = (x_{i1}^{[j]}, x_{i2}^{[j]}, \ldots, x_{i,n_j}^{[j]})^\top \in X_j,$$

is evaluated by using the buffer vector to complement the missing components. This is achieved by substituting the buffer's components that correspond to the contribution of the jth swarm, with the components of $x_i^{[j]}$ while retaining the rest of the buffer unchanged. Hence, the objective value assigned to $x_i^{[j]}$ is:

$$f_i^{[j]} = f(\mathcal{B}_i^{[j]}), \tag{6}$$

where,

$$\mathcal{B}_i^{[j]} = (\underbrace{b_1^{[1]}, \ldots, b_{n_1}^{[1]}}_{\text{unchanged}}, \ldots, \underbrace{x_{i1}^{[j]}, \ldots, x_{i,n_j}^{[j]}}_{\text{considered particle}}, \ldots, \underbrace{b_1^{[K]}, \ldots, b_{n_K}^{[K]}}_{\text{unchanged}})^{\top},$$

with $i = 1, 2, \ldots, N_j$, and $j = 1, 2, \ldots, K$.

A reasonable choice for the contributed information of each subswarm is its overall best position, i.e.:

$$b^{[k]} = p_g^{[k]} = \left(p_{g1}^{[k]}, p_{g2}^{[k]}, \ldots, p_{g,n_k}^{[k]}\right)^{\top},$$

for the kth subswarm. This choice produces a buffer that contains the overall best positions of all subswarms:

$$\mathcal{B} = (\underbrace{p_{g1}^{[1]}, \ldots, p_{g,n_1}^{[1]}}_{\text{overall best of } \mathcal{S}_1}, \underbrace{p_{g1}^{[2]}, \ldots, p_{g,n_2}^{[2]}}_{\text{overall best of } \mathcal{S}_2}, \ldots, \underbrace{p_{g1}^{[K]}, \ldots, p_{g,n_K}^{[K]}}_{\text{overall best of } \mathcal{S}_K})^{\top}. \quad (7)$$

By definition, this buffer constitutes the best position ever attained by the algorithm, i.e., it is the best obtained approximation of the global minimizer of $f(x)$.

Different approaches for building the buffer vector may result in different convergence properties of the algorithm. For example, instead of the overall best position, a randomly selected best position from each subswarm can be contributed to the buffer. Such a cooperative scheme is expected to exhibit slower convergence but higher diversity than the one described above. Thus, the user shall make a decision taking into consideration the desirable dynamics of the algorithm as well as possible information on the objective function, such as estimations of the number of local minima or their approximate distribution in the search space.

2.4. Cooperative micro-particle swarm optimization

Recently, a cooperative micro-PSO (COMPSO) algorithm was introduced [27]. This approach combines the flexibility of micro-PSO with the efficiency of cooperative approaches. The algorithm's operation is based on the same concepts as for the CPSO approach described in the previous section, i.e., it decomposes the original search space into low-dimensional subspaces, which are allocated to micro-PSO subswarms. The subswarms promote cooperation by sharing their findings, while their sizes are selected to be higher than the dimensions of the corresponding subspaces [27].

The subswarms shall assume very small size in micro-PSO. Thus, the dimension of the subspaces shall be selected such that the ratios $D_k = n_k/N_k$, $k = 1, 2, \ldots, K$, lie as close to zero as possible. In practice, a ratio value around 0.5 provides satisfactory results. For example, for subswarm size $N_k = 5$, a decomposition of the original search space into subspaces of dimension $n_k = 3$ for all k is a reasonable configuration, allocating a subswarm of 5 particles to each 3-dimensional subspace. Moreover, in order to avoid search stagnation and premature convergence, a lower bound of the standard deviation per direction component can be determined for each subswarm. If this threshold is violated then the corresponding subswarm is randomly restarted, although retaining its best positions [27].

A pseudocode of the COMPSO algorithm is provided in Table 1. It is worth noting that the buffer vector is updated immediately after the detection of a better position (steps 10–12 of the pseudocode) by copying the corresponding particle (which also becomes the overall best position for the corresponding subswarm) to the proper position in $\mathcal{B}$. Thus, the subsequent particles even of the same subswarm will be evaluated by using the updated buffer.

This is an *asynchronous* buffer update scheme that immediately exploits the discovered information, thereby producing higher overall convergence speed to the COMPSO algorithm. Alternatively, a *synchronous* scheme can be used where the buffer is updated after the evaluation of all particles and subswarms. Slower convergence speed shall be expected in this case.

Table 1
Pseudocode of the COMPSO algorithm [27].

Input	K (number of subswarms), N_k, n_k (subswarm's size and dimension), $k = 1, 2, \ldots, K$, $\mathcal{B}$ (buffer vector), $\sigma_{\min}$ (diversity threshold), $f(.)$ (objective function)
Step 1.	**Initialize** randomly all subswarms in their search spaces (subspaces of the original one).
Step 2.	**Initialize** buffer vector $\mathcal{B}$ using a randomly selected particle from each subswarm.
Step 3.	**Evaluate** all particles using $\mathcal{B}$ and set their best positions.
Step 4.	**While** (termination condition not met)
Step 5.	**Do** ($k = 1, 2, \ldots, K$)
Step 6.	**Do** ($i = 1, 2, \ldots, N_k$)
Step 7.	**Update** the particle $x_i^{[k]}$ using Eqs. (1) and (2).
Step 8.	**Evaluate** $x_i^{[k]}$ using Eq. (6) and the buffer $\mathcal{B}$.
Step 9.	**Update** the best position, $p_i^{[k]}$, using Eq. (3), and the buffer $\mathcal{B}$ using Eq. (7).
Step 10.	**If** ($f(x_i^{[k]}) < f(\mathcal{B})$) **Then**
Step 11.	**Copy** $x_i^{[k]}$ in the proper position of the buffer $\mathcal{B}$.
Step 12.	**End If**
Step 13.	**End Do**
Step 14.	**Compute** standard deviations, σ_j, $j = 1, 2, \ldots, n_k$, for all direction components.
Step 15.	**If** ($\min_j\{\sigma_j\} < \sigma_{\min}$) **Then**
Step 16.	**Re-initialize** the kth subswarm randomly, retaining its best positions.
Step 17.	**End If**
Step 18.	**End Do**
Step 19.	**End While**
Step 20.	**Print** buffer $\mathcal{B}$ and $f(\mathcal{B})$.

In [27] the performance of COMPSO was investigated on a set of widely used test problems for dimensions up to 1200. The results suggested that COMPSO can significantly outperform the standard PSO. In these experiments, the search space was *linearly* decomposed, i.e., the first 3 direction components were allocated to the first subswarm, the next 3 directions to the second subswarm, etc. Thus, the kth subswarm $\mathcal{S}_k$, $k = 1, 2, \ldots, K$, was operating on the coordinate directions, $3k-2$, $3k-1$, and $3k$, of the original objective function, occupying the corresponding positions in the buffer vector.

In the next section, the established master–slave model for parallel computing is briefly described.

2.5. Parallel master–slave model

The *master–slave* (MS) model is among the most popular approaches for parallel computing. Its popularity can be attributed to the straightforward exploitation of the parallelization capabilities of modern computer systems as well as to its easy implementation. The development of parallel MS variants of established algorithms usually requires minor programming effort and essential knowledge of the corresponding computer system. Also, usually it requires only minor alterations in the algorithm's structure. For these reasons, the MS model is considered as the first step in the transition of an algorithm from its serial to parallel implementation. Nevertheless, the MS model suffers disadvantages such as the sequential generation of slave processes and the heavy communication overheads imposed on the master processor [37,38].

In general, the MS model consists of one master process and M slave processes. Usually, the master is responsible for performing the main operations of the implemented algorithm, while the slaves are evoked whenever required to perform intermediate operations, in parallel. For instance, an MS model of an evolutionary algorithm

Table 2
Pseudocode for the synchronous (left) and asynchronous (right) evaluation procedure of N particles using $M < N$ slave processes.

Input	N (number of swarms), M (number of slave processes).	
	Synchronous evaluation	Asynchronous evaluation
Step 1.	**Set** $n \leftarrow 0$.	**Set** $n \leftarrow M$.
Step 2.	**While** $(n < N)$	**Send** M particles to slaves.
Step 3.	**Set** $k = \min\{M, N - n\}$.	**While** $(n < N)$
Step 4.	**Send** k particles to slaves.	**Wait** until a slave, s_i, finishes.
Step 5.	**Receive** the k objective values.	**Receive** the objective value.
Step 6.	**Update** algorithm parameters.	**Update** algorithm parameters.
Step 7.	**Set** $n \leftarrow n + k$.	**Set** $n \leftarrow n + 1$.
Step 8.	**End While**	**If** $(n < N)$ **Then**
Step 9.		**Send** a new particle to slave s_i.
Step 10.		**End If**
Step 11.		**End While**

would consist of a master process that retains the population and performs the main evolutionary operations, while the slaves would be evoked to concurrently evaluate the individuals. As can be easily comprehended, such a model would require only minor (if any) modifications to the algorithm's structure.

The expected gain of an MS parallel model is the significant reduction of the total wall-clock time required by the algorithm. In an environment of M slave processes the concurrent evaluation of M individuals is possible, reducing the total evaluation time of the population. Naturally, it is expected that the total gain of the parallel implementation with respect to the required time will be more significant in hard problem instances (e.g., in high-dimensional and time-consuming objective functions). On the other hand, it can be unaffected or even worsen in trivial problems of low-dimensionality and instantaneous function evaluations. In the latter case, it is usually better to stick to the serial implementation.

Parallel evolutionary algorithms (PEAs) have been frequently implemented within an MS framework. There are several comprehensive reference works that offer thorough analyses as well as current and future trends in PEAs research [39,40]. Regarding PSO, there is a number of existing parallel implementations [41], including significant applications such as nuclear engineering [42], reactive power dispatch [43], geotechnical analysis [44], data mining [45] as well as hardware implementations [46].

In the next section, the proposed master–slave model for the parallel COMPSO (PCOMPSO) approach is analyzed.

3. Master–slave model for parallel cooperative micro-particle swarm optimization

The COMPSO algorithm described in Section 2.4 is straightforwardly transferred in the context of a parallel MS model, resulting in the proposed Parallel COMPSO (PCOMPSO) approach. The master process is the main algorithmic module of the algorithm, retaining and updating all subswarms. On the other hand, the slaves are computational units evoked only for the evaluation of the particles. Thus, intense communication activity is expected between the master and the slave processes. The data that needs to be transmitted to a slave process for the evaluation of a particle is the particle itself and the buffer vector.

There are two possibilities for the evaluation of N particles in an MS environment with M slave processes:

(a) $M \geq N$, i.e., the number of available slave processes is larger than the number of particles. In this case, all particles can be concurrently evaluated.
(b) $M < N$, i.e., only M out of N particles can be concurrently evaluated.

In practice, the case (b) is met more frequently than (a) and the evaluation of the particles can be conducted in two alternative ways.

The first one assumes that a group of M particles are sent for evaluation from the master process to the slaves. After the evaluation of the whole group, the M objective values are returned to the master and another group of M particles is sent to the slaves. This is a type of *partially synchronous evaluation*, since a group of particles is sent for evaluation only after the whole previous group has finished. Alternatively, M particles can be initially sent for evaluation and wait until one objective value is returned to the master. Then, the master immediately sends another particle for evaluation to the idle slave process, regardless of the state of the rest. This is a type of *asynchronous evaluation* since each particle is individually set for evaluation as soon as there is a free slave process available. Pseudocode of the two evaluation schemes is given in Table 2.

The time efficiency of the synchronous or asynchronous evaluation scheme depends on the objective function. If the time required for a single function evaluation is almost identical for all points in the search space, then the synchronous evaluation model is expected to have minor (if any) differences with the asynchronous one with respect to the total evaluation time. On the other hand, if the time required for a single function evaluation varies significantly within the search space, the asynchronous model may be competitive to the synchronous one.

Besides the time cost of function evaluations, in parallel MS implementations there is an additional factor that affects the total wall-clock time of the algorithm. This is the communication overhead between the master and the slave processes. The synchronous and asynchronous evaluation schemes differ also in this feature. The synchronous model communicates new information (particles and buffer vector) from the master to the slave processes once per group of M particles.

On the other hand, the asynchronous model updates the buffer information after the receipt of each objective value from the slaves. Thus, during the evaluation of M particles the buffer can be updated up to M times. If this information is communicated immediately to all slaves, there is up to M times higher communication burden than in the synchronous model. Clearly, if the time required for a communication message is competitive to the average time spent for a single function evaluation then the asynchronous model may exhibit inferior time efficiency.

The serial COMPSO approach described in Table 1 is based on an asynchronous update scheme. The buffer and the best positions of each subswarm are allowed to update after each particle evaluation. This can accelerate the algorithm's convergence speed, since the new buffer (with the lowest function value attained so far) is immediately used for the evaluation of subsequent particles as derived from step 9 in Table 1.

Table 3
Pseudocode of the PCOMPSO approach. Operations performed by the slave processes appear in italics.

Input	K (number of subswarms), $\mathcal{B}$ (buffer vector), $\sigma_{\min}$ (diversity threshold), $f(.)$ (objective function), M (number of slave processes).
Step 1.	**Initialize** randomly all subswarms in their search spaces (subspaces of the original one).
Step 2.	**Initialize** buffer vector $\mathcal{B}$ using a randomly selected particle from each subswarm.
Step 3.	**Evaluate** all particles using $\mathcal{B}$ and set their best positions.
Step 4.	**While** (termination condition not met)
Step 5.	**While** (there are unevaluated particles)
Step 6.	**Build** buffer $\mathcal{B}$ using Eq. (7).
Step 7.	**Send** at most M particles and the buffer to the slaves.
Step 8.	***Evaluate*** *particles and return the M objective values to the master process.*
Step 9.	**Update** indices of best positions in the corresponding subswarms.
Step 10.	**End While**
Step 11.	**Do** ($k = 1, 2, \ldots, K$)
Step 12.	**Compute** the standard deviation, σ_i, for each direction component of the kth subswarm.
Step 13.	**Set** $\sigma_k \leftarrow \min_i\{\sigma_i\}$.
Step 14.	**If** ($\sigma_k < \sigma_{\min}$) **Then**
Step 15.	**Re-initialize** the kth subswarm, retaining its best positions.
Step 16.	**End If**
Step 17.	**End Do**
Step 18.	**If** (re-allocation is needed) **Then**
Step 19.	**Allocate** new direction components to subswarms and re-initialize them.
Step 20.	**End If**
Step 21.	**End While**
Step 22.	**Print** buffer $\mathcal{B}$ and $f(\mathcal{B})$.

However, in the parallel implementation such an update would require an excessive number of communication messages between the master and the slave processes, as explained above. For this reason, the partially synchronous model was adopted in PCOMPSO as exposed in the pseudocode of Table 3. Evidently, in PCOMPSO the buffer $\mathcal{B}$ is updated after the evaluation of the whole group of M particles, regardless of possibly better positions that may be found during the evaluation of the group.

Two points are worth noting in the pseudocode of PCOMPSO. The first one consists of the steps 14–16 in Table 3, which resemble the restarting procedure of COMPSO (see steps 15–17 in Table 1). These steps constitute an essential feature of micro–EAs to address the problem of premature convergence as mentioned in Section 2.4. The re-initialization is triggered as soon as a lower bound of the minimum standard deviation of the particles per direction is violated.

More specifically, if $\mathcal{S}_k$ is the kth subswarm with dimension n_k and size N_k, and $\sigma_{k,i}$, $i = 1, 2, \ldots, n_k$, are the standard deviations for the corresponding n_k direction components with $\sigma_{k,\min}$ denoting the smallest among them, then the condition:

$$\sigma_{\min} \leq \sigma_{k,\min}, \tag{8}$$

must hold, where $\sigma_{\min}$ is a user-defined parameter for the identification of search stagnation. If this condition is violated, the subswarm $\mathcal{S}_k$ is re-initialized within its subspace. However, its best positions are retained in order to exploit all the available information produced by the subswarm so far. The determination of a satisfactory value for $\sigma_{\min}$ depends on the problem at hand as well as on the desirable solution accuracy.

The second critical point of PCOMPSO consists of the steps 18–20 in Table 3. As previously mentioned, COMPSO employs a linear decomposition of the search space [27]. This scheme is the simplest straightforward approach for allocating direction components (subspaces) of the original problem to subswarms. However, it suffers a crucial drawback: it does not take into account the possibility of correlations among variables. Thus, if highly correlated variables are assigned to different subswarms, the algorithm's efficiency is expected to decline in terms of solution quality given a prespecified budget of function evaluations. This effect can be attributed to the algorithm's inability for rapid identification of patterns that dominate the variables' interactions and it has been observed also in single-swarm approaches. Nevertheless, it is intensified in multi-swarm variants such as the proposed one.

The aforementioned deficiency can be tackled through various techniques. Popular preprocessing approaches such as the application of principal components analysis may be well-analyzed and theoretically supported [47], but can also dramatically increase the computational burden of the algorithm due to demand for laborious linear algebra operations.

An alternative and computationally economic approach is the periodic *randomized re-allocation* of direction components to subswarms during the algorithm's execution. According to this approach, the subswarms are initialized based on linear decomposition but, after every t_{map} iterations, they are re-allocated to randomly selected direction components. Each re-mapping is accompanied by the re-initialization of the subswarms in the new subspaces.

This idea has been introduced as *random grouping* [48] and it has been applied on a CPSO model in [49]. If $\mathcal{S}_k$ is the kth subswarm, after every t_{map} iterations it is assigned to different direction components, $n_{r_1}, n_{r_2}, \ldots, n_{r_{n_k}}$, where the indices, $r_1, r_2, \ldots, r_{n_k}$, are randomly selected without replacement from the set $\{1, 2, \ldots, n\}$. Hence, a possibly unsuccessful assignment of direction components that would split highly correlated variables, will negatively affect the subswarms only for a limited number of iterations (controlled by t_{map}). This approach is implemented in steps 18–20 of the pseudocode in Table 3.

Finally, a comment of technical nature shall be made for step 7 in Table 3. The slave processes typically execute the following sequence of operations:

(1) **Receive** a particle and the buffer vector.
(2) **Compute** the objective value of the particle.
(3) **Send** the result to the master process.

If new data are sent to the slave processes during the second step (computation of the objective value), they are put in a waiting queue until the next "Receive" action of the slave. Thus, one may argue that the use of the phrase "at most" in step 7 of Table 3 can be replaced as follows:

Step 7. **Send** *all* particles and the buffer to the slaves.

However, this approach would produce two undesirable effects. The first one is that all N particles would be accompanied by the same buffer to the slaves. Thus, if a new buffer was produced during their evaluation, it would be unavailable to the forthcoming particles. The second effect stems from the fact that many parallel systems support waiting queues of rather limited size per slave process. Such systems may fail to retain in memory a large amount of messages for all particles. This can result in unpredictable behavior of the system, ranging from completely ignoring the redundant messages to hanging of the slave processes. For these reasons, it is recommended to use the original form of step 7 in Table 3.

In the next section, the experimentation environments for the assessment of PCOMPSO are described and results are reported for a widely used test suite.

4. Experimentation environments and parameter setup

PCOMPSO was applied on the widely used test problems reported in the Appendix of the present paper. The popularity of these problems is attributed to their different features, including unimodal, multimodal, separable and non-separable functions, straightforward generalization in arbitrary dimensions, as well as strongly and loosely correlated variables.

The algorithm was assessed over two different parallel computing hardware platforms, under different parameter settings. The details of their configuration are described in the following paragraphs.

4.1. Employed parallel computing systems

PCOMPSO was assessed over two parallel computing systems. The first system was one of the facilities of SHARCNET, which is a consortium of Canadian academic institutions that share a network of high-performance computers, operating under the umbrella of Compute/Calcul Canada. It consists of several core and specialized systems, supporting a plethora of tools for different applications. For the experiments conducted in the present paper, the *Saw* facility of SHARCNET was used. Saw consists of 336 Xeon 2.83 GHz nodes, each one offering 8 cores on 2 sockets and 16 GB of memory, summing to a total of 2688 CPUs. The system is devoted to parallel applications and supports InfiniBand host connections, while it runs under the XC 4.0 operating system.

Unfortunately, such systems are not frequently available, especially in non-academic environments. Hopefully, as already mentioned in Section 1, modern desktop computers with multicore processors can offer small-scale, shared-memory, parallel computing solutions to the user. Most important, such systems are widely spread since their cost is relatively low. This was the motivation for selecting as the second computer experimentation platform, a desktop PC consisting of an Intel® I7 2.66 GHz processor with 6 GB memory. The I7 processor has 4 cores and supports the Intel® Hyper-Threading technology. Running under the Ubuntu 9.10 64-bit Linux distribution, this machine offers 8 processing threads.

In both computer platforms, PCOMPSO was implemented in the C++programming language (GCC 4.4.1 compiler) using the MPI standard.

4.2. The parameter setup

For comparison purposes, the experimental setup used in [27] was mostly adopted in the present paper. The test problems were considered for dimensions $n = 300$, 600 and 1200. In both experimentation environments, the time required for a single function evaluation was measured to be of order $\mathcal{O}(10^{-6})$ second (i.e., 1 μs). Since parallel implementations have merit only in demanding time-consuming objective functions, a random time-delay was added to each function evaluation. The delay was defined as $d = \mathcal{R} \times d_{\max}$ seconds, where $\mathcal{R} \sim U([0, 1])$ is a random variable uniformly distributed in the range [0,1], and d differs for each function evaluation. The constant $d_{\max}$ was used to impose an upper bound on the time-delay. Two different levels of this parameter were considered, namely $d\max_1 = 10^{-4}$ and $d\max_2 = 10^{-3}$.

For each test problem, the original search space was decomposed in 3-dimensional subspaces undertaken by individual subswarms. As mentioned in Section 2.4, this setting is reasonable for micro-PSO approaches because it allows the use of small subswarms. The corresponding numbers of subswarms and particles per problem instance are reported in Table 4.

Both the linear and the random decomposition approach described in Section 3, were considered in the experiments. The lower bound of the diversity for the restarting condition in Eq. (8) was fixed to the value $\sigma_{\min} = 10^{-20}$. This is a reasonable value because allowing smaller diversities can rarely offer adequate local fine-tuning of the obtained solutions. Each experiment was terminated as soon as the algorithm reached a maximum number of $F_{\max} = k \times 10^3$ function evaluations, where k is the total number of particles per problem instance (given in last column of Table 4).

Table 4
Total number of particles and subswarms per problem dimension.

Dimension	Subswarms	Particles per subswarm	Total particles
300	100	5	500
600	200	5	1000
1200	400	5	2000

In the case of random problem decomposition, the frequency of re-allocation was set to the fixed value $t_{\text{map}} = 30$. For this choice, it was taken into consideration that as t_{map} approaches the value 10^3, the impact of the random scheme on the algorithm's performance approaches that of the linear scheme. Hence, a remarkably small value was expected to reveal the differences of the two schemes with more clarity. Besides that, preliminary experiments suggested that small deviations from this value were not producing significantly different results.

Regarding the PSO parameters, the default set defined in Eq. (4) for the constriction coefficient PSO variant was adopted in all subswarms. All the aforementioned parameters are summarized in Table 5. Moreover, the PCOMPSO algorithm was tested using different numbers of CPUs. For the Saw cluster of the SHARCNET network, there were used:

$$N_{\text{cpu}}^{(\text{SH})} = 1, 3, 9, 17, 25, 33,$$

CPUs, with the value 1 corresponding to the serial variant, i.e., the standard COMPSO. For the I7 system, less CPUs were available:

$$N_{\text{cpu}}^{(\text{I7})} = 1, 5, 8.$$

It must be underlined that these values refer to the total number of CPUs, i.e., including the master. The number of slave processes is obtained simply by subtracting 1 from the aforementioned values (except from the serial variant).

For each algorithm and problem instance 30 independent experiments were performed. The average performance of the algorithms was derived through statistical analysis of the results. For the serial COMPSO approach, the same total number of function evaluations with the PCOMPSO approaches was assumed in all problem instances. Also, its swarm size was equal to the total number of particles used in the corresponding PCOMPSO approaches. Hence, all compared approaches were given the same computational budget as anticipated in a fair comparison framework.

4.3. Performance measures

The performance was assessed with three measures. The first measure was the *quality of solution*, i.e., the value of the best detected solution within the allowed number of function

Table 5
Parameter values for PCOMPSO.

Parameter	Description	Value
$\sigma_{\min}$	Diversity threshold	10^{-20}
$F_{\max}$	Maximum function evaluations	$k \times 10^3$ (k: number of particles)
χ	Constriction coefficient	0.729
c_1, c_2	Cognitive and social parameter	2.05
NT	Neighborhood topology	Ring
r	Neighborhood radius	1

Table 6
Results for TP0 on the academic cluster.

Dim.		Number of CPUs					
		1 (COMPSO)	3	9	17	25	33
300	μ_f	**8.93e−26**	2.09e−25	4.61e−25	4.58e−25	1.97e−24	4.88e−24
(Lin)	σ_f	**2.44e−25**	1.06e−24	1.70e−24	1.38e−24	7.02e−24	1.71e−23
	$d_{\max_1}$	26.80 (0.20)	18.67 (0.03)	7.22 (0.01)	4.53 (0.01)	3.58 (0.02)	3.18 (0.03)
	$d_{\max_2}$	252.61 (0.37)	169.24 (0.13)	58.06 (0.03)	31.70 (0.02)	21.83 (0.03)	17.20 (0.07)
	cssd	25, 33	17, 25, 33	25, 33	3	1, 3, 9	1, 3, 9
	μ_f	8.93e−26	**4.34e−30**	1.18e−29	2.74e−29	3.72e−29	1.09e−28
(Rand)	σ_f	2.44e−25	**2.72e−30**	7.38e−30	1.87e−29	2.18e−29	7.16e−29
	$d_{\max_1}$	26.80 (0.20)	18.70 (0.04)	7.23 (0.05)	4.52 (0.01)	3.57 (0.02)	3.15 (0.04)
	$d_{\max_2}$	252.61 (0.37)	169.19 (0.13)	58.06 (0.04)	31.70 (0.02)	21.80 (0.05)	17.23 (0.06)
	cssd	All	All	All	All	All	All
600	μ_f	9.52e−22	**7.17e−23**	1.34e−19	4.23e−22	5.02e−20	7.69e−22
(Lin)	σ_f	4.88e−21	**1.93e−22**	7.33e−19	1.33e−21	1.85e−19	2.36e−21
	$d_{\max_1}$	54.26 (0.04)	39.18 (0.06)	15.47 (0.03)	10.36 (0.03)	8.59 (0.04)	7.85 (0.04)
	$d_{\max_2}$	506.63 (0.30)	340.26 (0.18)	114.92 (0.05)	63.78 (0.07)	44.83 (0.06)	35.57 (0.07)
	cssd	17, 25, 33	9, 17, 25, 33	3, 25, 33	1, 3	1, 3, 9	1, 3, 9
	μ_f	9.52e−22	**1.29e−26**	2.36e−26	3.78e−26	5.32e−26	8.48e−26
(Rand)	σ_f	4.88e−21	**7.80e−27**	8.17e−27	1.42e−26	2.62e−26	4.10e−26
	$d_{\max_1}$	54.26 (0.04)	38.86 (0.07)	15.44 (0.04)	10.35 (0.03)	8.61 (0.03)	7.85 (0.05)
	$d_{\max_2}$	506.63 (0.30)	340.00 (0.17)	114.83 (0.06)	63.85 (0.03)	44.89 (0.06)	35.51 (0.08)
	cssd	All	All	All	All	All	All
1200	μ_f	1.44e−19	1.48e−19	**1.71e−20**	9.50e−18	4.73e−19	4.46e−16
(Lin)	σ_f	7.16e−19	5.55e−19	**2.17e−20**	5.18e−17	1.98e−18	2.44e−15
	$d_{\max_1}$	116.89 (0.38)	85.58 (0.20)	37.49 (0.10)	27.28 (0.05)	23.98 (0.08)	22.85 (0.12)
	$d_{\max_2}$	1018.74 (2.04)	686.95 (0.27)	234.87 (0.08)	130.24 (0.07)	94.96 (0.10)	76.31 (0.13)
	cssd	25, 33	9, 17, 25, 33	3, 33	3	1, 3	1, 3, 9
	μ_f	1.44e−19	**2.52e−24**	5.42e−24	7.30e−24	7.14e−24	1.03e−23
(Rand)	σ_f	7.16e−19	**1.05e−24**	1.92e−24	2.77e−24	2.11e−24	3.85e−24
	$d_{\max_1}$	116.89 (0.38)	84.91 (0.18)	37.32 (0.07)	27.22 (0.05)	24.33 (0.12)	22.70 (0.09)
	$d_{\max_2}$	1018.74 (2.04)	687.04 (0.33)	234.68 (0.09)	130.32 (0.12)	94.86 (0.09)	76.32 (0.11)
	cssd	All	All	All	1, 3, 9, 33	1, 3, 9, 33	All

evaluations. This measure does not take into consideration the required wall-clock time, i.e., it is not explicitly affected by the parallel implementation. However, it may be implicitly affected, since the parallel model presents a new buffer vector to the slaves only after the evaluation of M particles instead of every particle in the serial model. The quality of solutions per algorithm is measured with the mean and standard deviation, μ_f and σ_f, respectively, of the obtained solution values averaged over the number of experiments.

In order to confirm statistically significant differences between the considered variants, Wilcoxon rank sum tests were conducted for each pair (i, j) of approaches, where i and j denote the corresponding numbers of CPUs, namely $i, j \in \{1, 3, 9, 17, 25, 33\}$ (for the Saw system) or $i, j \in \{1, 5, 8\}$ (for the desktop system) with $i \neq j$. The null hypothesis of the rank sum tests was equality of medians between the compared samples and its validity was tested at a 95% significance level.

In addition to solution quality, the wall-clock time was recorded at each experiment and its mean and standard deviation, averaged over all experiments, were computed. Using these values, the second and the third performance measures were computed. These measures are related to the parallel implementation and they are the *speedup*, defined as:

$$s_M = \frac{E(T_1)}{E(T_M)},$$

and the *time efficiency*, defined as:

$$e_M = \frac{s_M}{M},$$

where M denotes the number of the employed CPUs; $E(T_1)$ is the expected required time of the algorithm when executed on a single CPU; and $E(T_M)$ is the expected required time when M CPUs are used. The expectations are approximated by the mean values of the corresponding wall-clock time averaged over all experiments (recall that 30 experiments were performed per case).

5. Experimental results

For better presentation, the obtained results for the two experimentation environments are reported in separate sections, starting with the case of the Saw (SHARCNET) academic cluster.

5.1. Results for the academic cluster

The PCOMPSO approach was tested using 1 (serial COMPSO), 3, 9, 17, 25 and 33 CPUs on the Saw facility of the SHARCNET academic cluster grid. As previously mentioned, this is the total number of CPUs, i.e., including the master process. The results are reported in Tables 6–10 and graphically illustrated in Figs. 1–5. The linear decomposition scheme is denoted as "Lin", while the random one is denoted as "Rand".

Each table reports for each dimension and decomposition scheme, the mean μ_f and the standard deviation σ_f of the best solution values, averaged over the 30 experiments. Also, it reports the mean and the standard deviation (in parenthesis) of the required wall-clock time for the two time-delay levels, $d\max_1$ and $d\max_2$. Finally, it reports the comparisons with the rest cases where statistical significant differences in solution quality were detected (this is denoted as "cssd" in the tables). The best solution (in terms of function value) per problem instance and decomposition scheme is boldfaced.

Table 7
Results for TP1 on the academic cluster.

Dim.		Number of CPUs					
		1 (COMPSO)	3	9	17	25	33
300	μ_f	**3.51e+02**	3.52e+02	3.60e+02	3.66e+02	3.71e+02	4.18e+02
(Lin)	σ_f	**5.83e+01**	4.73e+01	3.94e+01	6.25e+01	6.79e+01	1.27e+02
	$d_{\max_1}$	27.04 (0.10)	18.75 (0.03)	7.25 (0.02)	4.53 (0.01)	3.59 (0.02)	3.16 (0.03)
	$d_{\max_2}$	253.18 (0.47)	169.25 (0.14)	58.03 (0.03)	31.70 (0.02)	21.83 (0.04)	17.23 (0.07)
	cssd	33	33	33	33	none	none
	μ_f	**3.51e+02**	9.64e+02	9.88e+02	1.04e+03	1.09e+03	9.36e+02
(Rand)	σ_f	**5.83e+01**	6.68e+02	7.66e+02	8.66e+02	7.72e+02	4.94e+02
	$d_{\max_1}$	27.04 (0.10)	18.74 (0.03)	7.23 (0.02)	4.52 (0.01)	3.58 (0.02)	3.15 (0.02)
	$d_{\max_2}$	253.18 (0.47)	169.25 (0.15)	58.01 (0.03)	31.70 (0.03)	21.81 (0.04)	17.21 (0.05)
	cssd	all	1	1	1	1	1
600	μ_f	7.34e+02	**7.27e+02**	7.54e+02	7.78e+02	7.66e+02	7.84e+02
(Lin)	σ_f	6.38e+01	**7.17e+01**	9.58e+01	9.00e+01	7.09e+01	8.70e+01
	$d_{\max_1}$	55.61 (0.26)	39.46 (0.07)	15.52 (0.03)	10.42 (0.02)	8.59 (0.03)	7.95 (0.05)
	$d_{\max_2}$	508.37 (1.70)	340.45 (0.17)	114.86 (0.06)	63.82 (0.03)	44.85 (0.07)	35.58 (0.09)
	cssd	17, 33	17, 25, 33	None	1, 3	3	1, 3
	μ_f	**7.34e+02**	1.63e+03	1.67e+03	1.77e+03	1.70e+03	1.60e+03
(Rand)	σ_f	**6.38e+01**	6.82e+02	5.60e+02	7.32e+02	4.35e+02	2.51e+02
	$d_{\max_1}$	55.61 (0.26)	39.18 (0.07)	15.52 (0.04)	10.38 (0.03)	8.62 (0.04)	7.84 (0.04)
	$d_{\max_2}$	508.37 (1.70)	340.12 (0.17)	114.82 (0.05)	63.81 (0.03)	44.86 (0.08)	35.61 (0.08)
	cssd	All	1, 17, 33	1	1	1	1
1200	μ_f	1.52e+03	1.54e+03	**1.52e+03**	1.58e+03	1.57e+03	1.57e+03
(Lin)	σ_f	1.06e+02	1.27e+02	**9.90e+01**	1.32e+02	1.05e+02	1.05e+02
	$d_{\max_1}$	119.48 (0.87)	86.35 (0.18)	37.83 (0.10)	27.40 (0.06)	24.24 (0.09)	22.87 (0.07)
	$d_{\max_2}$	1023.38 (1.18)	688.59 (0.27)	234.99 (0.08)	130.33 (0.06)	95.04 (0.07)	76.54 (0.10)
	cssd	None	None	25	None	9	None
	μ_f	**1.52e+03**	3.00e+03	2.96e+03	3.04e+03	3.20e+03	3.33e+03
(Rand)	σ_f	**1.06e+02**	3.98e+02	5.19e+02	4.74e+02	7.95e+02	1.32e+03
	$d_{\max_1}$	119.48 (0.87)	85.92 (0.16)	37.59 (0.06)	27.37 (0.06)	23.97 (0.02)	22.78 (0.07)
	$d_{\max_2}$	1023.38 (1.18)	688.05 (0.30)	234.89 (0.18)	130.33 (0.13)	95.01 (0.08)	76.40 (0.17)
	cssd	All	1	1	1	1	1

For example, in Table 6, for the 300-dimensional instance of TP0 and for the linear decomposition scheme, the serial COMPSO approach (1 CPU) achieved average solution value $\mu_f = 8.93e-26$ with standard deviation $\sigma_f = 2.44e-25$ in the 30 experiments. Its average wall-clock time for the time-delay level $d_{\max_1}$ was equal to 26.80 seconds with standard deviation 0.20, while for $d_{\max_2}$ the corresponding quantities were equal to 252.61 and 0.37, respectively. Finally, its solution quality exhibited statistically significant differences only with the cases of 25 and 33 CPUs.

It shall be noted that the single-CPU case corresponds to the original COMPSO approach, which does not use re-allocation of the problem's direction components. Thus, the "Lin" and "Rand" entries for 1 CPU are identical in the tables. The random number generator was initialized with the same seed for each algorithm, in order to avoid possible performance biasing due to differences between pseudo–random number sequences or instabilities of the generator.

The figures consist of an upper part (line plots) that illustrates the speedup and a lower part (bar graphs) that illustrates time efficiency of PCOMPSO. Also, the figures on the left column correspond to the time-delay level $d_{\max_1} = 10^{-4}$, while figures on the right column correspond to $d_{\max_2} = 10^{-3}$. For each graph type (line plot or bar graph), the first two figures (first row) correspond to the "Lin" decomposition scheme, while the rest (second row) correspond to the "Rand" decomposition scheme.

The results for the unimodal problem TP0 are reported in Table 6. In the 300-dimensional case with linear decomposition, the serial COMPSO outperforms all parallel approaches in terms of solution quality. However, this superiority is statistically significant only in the cases of 25 and 33 CPUs. For the rest of the cases, the performance of COMPSO is statistically equivalent with that of PCOMPSO.

The nice performance of COMPSO should be anticipated in TP0, if we take into consideration two factors: the unimodality of the specific problem and the asynchronous update scheme for the particles in COMPSO, contrary to the partially synchronous update scheme of PCOMPSO. As described in Section 3, the asynchronous update can produce faster convergence although it fosters the danger of rapid loss of diversity and search stagnation. Nevertheless, this weakness can be negligible in unimodal cases due to the lack of multiple stationary points.

The partially synchronous update of PCOMPSO explains also its declining performance in TP0 as the number of CPUs increases. Indeed, a parallel approach with 3 CPUs may update its buffer after every 2 function evaluations, while an approach with 33 CPUs may update the buffer after every 32 function evaluations. For this reason, we observe in Table 6 that the parallel approaches with less CPUs exhibit superior performance with respect to solution quality.

For the 600- and 1200-dimensional instances of TP0 with linear decomposition, the parallel approaches with small number of CPUs outperform COMPSO. However, similarly to the 300-dimensional case, this is not accompanied by statistical significance. The picture is different when the random decomposition scheme is used. In all experiments with this scheme, the solutions of PCOMPSO were 4 to 5 orders of magnitude better than these of the serial COMPSO, regardless of the problem's dimension.

This may seem odd, since TP0 consists of a separable objective function; therefore re-allocation of the direction components should not substantially affect the algorithm's output. This is

Table 8
Results for TP2 on the academic cluster.

Dim.		Number of CPUs					
		1 (COMPSO)	3	9	17	25	33
300	μ_f	3.38e+02	3.43e+02	3.53e+02	3.43e+02	**3.30e+02**	3.39e+02
(Lin)	σ_f	2.85e+01	3.22e+01	3.60e+01	3.65e+01	**3.16e+01**	2.73e+01
	$d_{\max_1}$	33.29 (0.30)	19.33 (0.03)	7.41 (0.01)	4.61 (0.01)	3.61 (0.02)	3.18 (0.04)
	$d_{\max_2}$	260.41 (4.92)	169.85 (0.14)	58.18 (0.03)	31.77 (0.03)	21.87 (0.03)	17.30 (0.05)
	cssd	None	None	25	None	9	None
	μ_f	3.38e+02	1.45e+01	1.32e+01	**1.28e+01**	1.55e+01	1.57e+01
(Rand)	σ_f	2.85e+01	8.63e+00	9.79e+00	**6.01e+00**	8.06e+00	9.66e+00
	$d_{\max_1}$	33.29 (0.30)	19.34 (0.04)	7.38 (0.02)	4.60 (0.01)	3.64 (0.03)	3.20 (0.05)
	$d_{\max_2}$	260.41 (4.92)	169.86 (0.14)	58.21 (0.03)	31.76 (0.04)	21.80 (0.06)	17.29 (0.06)
	cssd	All	1	1	1	1	1
600	μ_f	**6.59e+02**	6.61e+02	6.68e+02	6.65e+02	6.68e+02	6.80e+02
(Lin)	σ_f	**3.65e+01**	4.17e+01	3.77e+01	4.60e+01	3.61e+01	4.34e+01
	$d_{\max_1}$	79.67 (0.40)	41.76 (0.07)	16.51 (0.03)	10.68 (0.02)	8.80 (0.04)	8.02 (0.04)
	$d_{\max_2}$	532.07 (0.51)	342.76 (0.17)	115.58 (0.05)	64.08 (0.06)	45.00 (0.06)	35.81 (0.06)
	cssd	None	None	None	None	None	None
	μ_f	6.59e+02	3.54e+01	3.03e+01	3.58e+01	**2.99e+01**	3.47e+01
(Rand)	σ_f	3.65e+01	1.02e+01	1.14e+01	1.16e+01	**1.27e+01**	1.11e+01
	$d_{\max_1}$	79.67 (0.40)	41.54 (0.06)	16.07 (0.03)	10.68 (0.02)	8.80 (0.01)	8.02 (0.07)
	$d_{\max_2}$	532.07 (0.51)	342.56 (0.18)	115.45 (0.05)	64.15 (0.04)	45.04 (0.08)	35.76 (0.05)
	cssd	All	1, 25	1	1, 25	1, 3, 17, 33	25
1200	μ_f	1.25e+03	**1.23e+03**	1.25e+03	1.25e+03	1.27e+03	1.24e+03
(Lin)	σ_f	4.04e+01	**4.74e+01**	7.05e+01	4.55e+01	5.38e+01	6.10e+01
	$d_{\max_1}$	219.09 (0.95)	95.89 (0.19)	39.96 (0.09)	28.59 (0.05)	24.98 (0.04)	23.41 (0.09)
	$d_{\max_2}$	1124.17 (1.69)	698.11 (0.27)	237.13 (0.11)	131.63 (0.11)	96.02 (0.16)	77.08 (0.13)
	cssd	None	25	None	None	3, 33	25
	μ_f	1.25e+03	7.84e+01	7.73e+01	7.67e+01	7.90e+01	**7.56e+01**
(Rand)	σ_f	4.04e+01	1.55e+01	1.52e+01	2.08e+01	1.77e+01	**2.07e+01**
	$d_{\max_1}$	219.09 (0.95)	95.42 (0.17)	40.00 (0.06)	31.18 (14.12)	25.00 (0.04)	23.38 (0.10)
	$d_{\max_2}$	1124.17 (1.69)	697.28 (0.30)	237.09 (0.08)	131.53 (0.06)	95.77 (0.13)	76.93 (0.09)
	cssd	All	1	1	1	1	1

true, however it does not take into consideration a detail of the re-allocation procedure: all subswarms are restarted after their assignment to new direction components. The restarting enhances the exploration capability of the subswarms, while the best detected solutions are retained in the buffer. This produces a performance boost for the parallel approaches, in contrast to the standard COMPSO that uses only linear decomposition.

In almost all experiments with the random decomposition scheme, the results were statistically significant as reported in Table 6. Also, the performance of the parallel approaches is reduced as the number of CPUs increases. This effect can be attributed to the same reasons mentioned above for the linear decomposition case, i.e., the partially synchronous update scheme of PCOMPSO. Summarizing the results for TP0 with respect to solution quality, we can infer that it is a quite trivial problem easily tackled either with the serial COMPSO or the PCOMPSO approach. The serial approach can offer satisfactory solutions in low–dimensional problem instances. On the other hand, the parallel approaches can attain better solutions in higher dimensions, especially when a relatively small number of CPUs is used in combination with the random decomposition scheme.

Regarding the required wall–clock time of the algorithms in TP0, we can make some interesting observations in Table 6. First, we notice that the two decomposition schemes require almost identical execution time. This was expected since exactly the same number of function evaluations was allowed for both decomposition schemes. The minor time differences can be mostly attributed to fluctuations of the system's load as well as to the slight additional time-requirements of the re-allocation procedure of the random decomposition scheme.

Fig. 1 illustrates the obtained speedup and time efficiency for TP0. Clearly, the gain for both time-performance measures is considerably higher when the time-delay level is high. Also, increasing the problem's dimension reduces the speedup. This was anticipated, since the manipulation and transmission of higher-dimensional vectors imposes heavier computation and communication burden on the parallel algorithms. Concluding, the experimental evidence indicate that in simple (convex) problems such as TP0, there is no need to employ a high number of CPUs in PCOMPSO unless significant time-delays are expected in the function evaluations.

The next considered test problem, TP1, is the generalized Rosenbrock function. Its 2-dimensional instance is considered to be a rather simple problem for swarm intelligence algorithms. However, its high-dimensional instances are considered to be significantly harder problems with numerous stationary points, while there are also correlations between adjacent variables.

The results for TP1 are reported in Table 7 and the time–performance measures are illustrated in Fig. 2. In general, we observe that the linear decomposition scheme offers better results than the random one for both COMPSO and PCOMPSO. Its superiority can be ascribed to the aforementioned correlations between adjacent variables. Indeed, linear decomposition allocates a group of subsequent direction components to each subswarm. Therefore, for each subswarm, correlations mostly exist between its allocated direction components. This allows the efficient capturing of the problem's dynamics per group of subsequent direction components, as well as the easier coordination of the subswarms towards the global minimizer. Additionally, we observe that in most cases the performance of COMPSO was

Table 9
Results for TP3 on the academic cluster.

Dim.		Number of CPUs					
		1 (COMPSO)	3	9	17	25	33
300	μ_f	6.32e−02	8.89e−02	8.44e−02	**1.36e−01**	4.75e−02	4.96e−02
(Lin)	σ_f	1.31e−01	1.73e−01	1.40e−01	**2.23e−01**	9.61e−02	1.02e−01
	$d_{\max_1}$	31.92 (0.12)	19.84 (0.03)	7.51 (0.02)	4.67 (0.01)	3.68 (0.02)	3.23 (0.03)
	$d_{\max_2}$	257.78 (0.32)	170.36 (0.13)	58.36 (0.03)	31.82 (0.03)	21.89 (0.04)	17.34 (0.06)
	cssd	None	None	25	25, 33	9, 17	17
	μ_f	6.32e−02	**1.18e−02**	6.23e−03	6.19e−03	1.55e−03	7.73e−03
(Rand)	σ_f	1.31e−01	**4.18e−02**	1.42e−02	1.50e−02	5.11$e-03$	1.78e−02
	$d_{\max_1}$	31.92 (0.12)	19.85 (0.04)	7.50 (0.01)	4.67 (0.01)	3.69 (0.02)	3.23 (0.03)
	$d_{\max_2}$	257.78 (0.32)	170.34 (0.13)	58.34 (0.02)	33.03 (1.53)	21.87 (0.03)	17.32 (0.05)
	cssd	all	1	1	1	1	1
600	μ_f	6.61e−02	9.30e−02	7.90e−02	**1.38e−01**	8.15e−02	5.56e−02
(Lin)	σ_f	1.45e−01	2.39e−01	1.71e−01	**2.52e−01**	1.30e−01	1.95e−01
	$d_{\max_1}$	75.73 (0.33)	43.81 (0.06)	16.67 (0.03)	10.95 (0.02)	9.00 (0.03)	8.15 (0.08)
	$d_{\max_2}$	527.01 (0.63)	344.74 (0.21)	116.09 (0.06)	64.37 (0.03)	45.13 (0.07)	35.91 (0.08)
	cssd	None	None	None	None	None	None
	μ_f	**6.61e−02**	3.04e−03	6.03e−03	2.39e−03	2.07e−03	6.82e−03
(Rand)	σ_f	**1.45e−01**	7.71e−03	1.51e−02	4.59e−03	6.14e−03	1.58e−02
	$d_{\max_1}$	75.73 (0.33)	43.56 (0.05)	16.58 (0.04)	10.94 (0.02)	9.04 (0.03)	8.15 (0.06)
	$d_{\max_2}$	527.01 (0.63)	344.59 (0.17)	116.02 (0.05)	67.74 (2.01)	45.10 (0.08)	35.90 (0.08)
	cssd	All	1	1	1	1, 33	1, 25
1200	μ_f	4.06e−02	7.75e−02	9.91e−02	7.27e−02	7.41e−02	**1.36e−01**
(Lin)	σ_f	5.97e−02	2.06e−01	2.05e−01	2.07e−01	1.39e−01	**3.06e−01**
	$d_{\max_1}$	197.40 (0.84)	103.87 (0.19)	42.14 (0.04)	29.57 (0.05)	25.83 (0.08)	23.93 (0.08)
	$d_{\max_2}$	1100.65 (1.13)	705.92 (0.33)	337.12 (342.66)	132.56 (0.09)	96.52 (0.12)	77.64 (0.14)
	cssd	None	None	None	None	None	None
	μ_f	4.06e−02	3.61e−03	1.49e−02	6.90e−03	6.84e−03	**1.18e−02**
(Rand)	σ_f	5.97e−02	9.78e−03	6.61e−02	1.14e−02	1.05e−02	**5.01e−02**
	$d_{\max_1}$	197.40 (0.84)	103.66 (0.14)	42.05 (0.07)	29.68 (0.05)	25.63 (0.10)	23.98 (0.08)
	$d_{\max_2}$	1100.65 (1.13)	707.34 (0.28)	239.14 (0.14)	132.53 (0.15)	96.61 (0.14)	77.42 (0.11)
	cssd	All	1	1	1	1	1

statistically equivalent with that of PCOMPSO as reported in Table 7.

Regarding the wall-clock time required by PCOMPSO, we can make similar observations with TP0. The time-performance gain is increased for the highest level of time-delay, while increasing the problem's dimension results in inferior values for the speedup. As a conclusion for TP1 we can state that, under the linear decomposition scheme, PCOMPSO and COMPSO can produce statistically equivalent results with respect to solution quality; although, PCOMPSO can achieve this performance significantly faster with respect to the required wall-clock time.

The next problem, TP2, has a plethora of local and a single global minimizer within the search space. The results for this problem are reported in Table 8 and time-related performance is illustrated in Fig. 3. As a first observation, we notice that the wall-clock time required by the serial COMPSO is increased compared to the previous test problems. This increase may seem odd since the total number of function evaluations was unchanged. However, it can be explained by the defining function of TP2, which involves the $\cos(x)$ trigonometric function. Such functions are approximately computed in the algorithm, requiring some additional time than the squared sums of the previous test problems. Therefore, besides the random time-delay imposed on each function evaluation, TP2 was inherently more time-consuming than TP0 and TP1 especially in its high-dimensional instances.

Regarding the quality of solutions, there is a clear superiority of the PCOMPSO approaches with the random decomposition scheme, against all approaches with the linear decomposition scheme. Comparing only approaches of same decomposition, there is no statistically significant difference in most cases as we can see in Table 8. Regarding the time-performance in TP2, we can notice a different pattern than in previous problems. As illustrated in Fig. 3, the speedup and time efficiency for the 1200-dimensional instances of TP2 is superior to that of lower-dimensional instances when a small number of CPUs is used. However, this superiority vanishes as the numbers of CPUs increases obviously due to the heavier communication overhead between the master and the slave processes. This indicates that in cases of complex high-dimensional problems, PCOMPSO can produce superior results than COMPSO, especially under the random decomposition scheme. Besides that, it can attain a significant gain in wall-clock time when a relatively small number of CPUs is used, similarly to TP0.

The results for the test problems TP3 and TP4 confirm the aforementioned observations. TP3 was proved to be an easier problem than TP2, as implied by the average quality of the obtained solutions reported in Table 9. A common property of TP2 and TP3 is the existence of terms that are approximately computed in their defining objective functions. This explains the similar wall-clock time reported in Tables 8 and 9. Also, PCOMPSO with the random decomposition scheme achieves increasingly better performance as the problem's dimension increases, when a small number of CPUs is used. This is illustrated also in Fig. 4. Moreover, we can observe that this effect becomes more intense for the lower time-delay level ($d_{\max_1}$). This is a consequence of the magnitude of the induced time-delay, which becomes comparable to the time required for the evaluation of the objective function.

Similar observations can be made for TP4. The results are reported in Table 10 and time–performance is illustrated in Fig. 5. In addition, the quality of the obtained solutions for TP4 implies that the steep funnels of the objective function around the global minimizer promote the detection of better solutions than in the previous test problems, regardless of the huge number of local minima.

Table 10
Results for TP4 on the academic cluster.

Dim.		Number of CPUs					
		1 (COMPSO)	3	9	17	25	33
300	μ_f	4.00e−13	4.11e−13	4.17e−02	3.99e−13	**3.84e−13**	4.54e−02
(Lin)	σ_f	4.65e−14	8.69e−14	2.28e−01	6.18e−14	**4.15e−14**	2.48e−01
	$d_{\max_1}$	29.35 (1.18)	19.30 (0.03)	7.38 (0.02)	4.63 (0.01)	3.63 (0.02)	3.20 (0.03)
	$d_{\max_2}$	255.33 (0.49)	169.81 (0.12)	58.23 (0.03)	31.78 (0.02)	21.90 (0.05)	17.29 (0.05)
	cssd	None	None	None	None	33	25
	μ_f	4.00e−13	2.62e−13	**2.59e−13**	2.61e−13	2.68e−13	2.70e−13
(Rand)	σ_f	4.65e−14	1.85e−14	**1.76e−14**	1.68e−14	1.74e−14	1.89e−14
	$d_{\max_1}$	29.35 (1.18)	19.33 (0.03)	7.39 (0.01)	4.64 (0.01)	3.64 (0.02)	3.20 (0.04)
	$d_{\max_2}$	255.33 (0.49)	169.79 (0.14)	58.17 (0.05)	31.76 (0.02)	21.88 (0.05)	17.29 (0.07)
	cssd	all	1	1, 33	1	1	1, 9
600	μ_f	4.26e−02	1.74e−12	1.43e−12	**1.11e−12**	8.07e−12	1.68e−12
(Lin)	σ_f	2.33e−01	3.96e−12	9.61e−13	**3.92e−13**	3.46e−11	9.85e−13
	$d_{\max_1}$	65.39 (1.14)	41.55 (0.05)	16.08 (0.03)	10.67 (0.03)	8.73 (0.01)	8.05 (0.03)
	$d_{\max_2}$	517.79 (1.11)	343.62 (0.18)	115.55 (0.05)	64.10 (0.03)	45.10 (0.06)	36.83 (1.77)
	cssd	25, 33	33	None	33	1	1, 3, 17
	μ_f	4.26e−02	**5.60e−13**	5.68e−13	5.63e−13	5.81e−13	5.71e−13
(Rand)	σ_f	2.33e−01	**2.52e−14**	2.95e−14	3.16e−14	3.23e−14	3.18e−14
	$d_{\max_1}$	65.39 (1.14)	41.62 (0.05)	16.03 (0.04)	10.71 (0.02)	8.78 (0.02)	8.06 (0.06)
	$d_{\max_2}$	517.79 (1.11)	342.39 (0.17)	115.61 (0.04)	64.12 (0.03)	45.00 (0.06)	35.80 (0.06)
	cssd	all	1, 25	1	1	1, 3	1
1200	μ_f	**1.15e−11**	1.22e−11	1.50e−11	8.62e−12	1.93e−11	1.47e−11
(Lin)	σ_f	**1.58e−11**	1.29e−11	1.81e−11	4.14e−12	3.90e−11	1.44e−11
	$d_{\max_1}$	163.84 (3.55)	95.38 (0.15)	39.92 (0.06)	28.52 (0.05)	24.88 (0.06)	23.47 (0.09)
	$d_{\max_2}$	1068.61 (4.32)	697.33 (0.27)	237.10 (0.08)	131.49 (0.06)	95.43 (0.11)	76.99 (0.10)
	cssd	25, 33	none	none	25, 33	1, 17	1, 17
	μ_f	**1.15e−11**	1.18e−12	1.23e−12	1.22e−12	1.26e−12	1.25e−12
(Rand)	σ_f	**1.58e−11**	3.98e−14	4.03e−14	4.18e−14	4.85e−14	4.84e−14
	$d_{\max_1}$	163.84 (3.55)	95.81 (0.15)	39.84 (0.05)	28.58 (0.07)	24.91 (0.05)	23.42 (0.13)
	$d_{\max_2}$	1068.61 (4.32)	696.90 (0.30)	237.22 (0.08)	131.53 (0.09)	95.60 (0.12)	77.08 (0.11)
	cssd	All	All	1, 3, 25	1, 3, 25, 33	1, 3, 9, 17	1, 3, 17

Summarizing the gained experience from the application of the master–slave PCOMPSO model on the Saw facility of the SHARCNET grid, we can state the following remarks:

(a) Overall, the PCOMPSO approach with the random decomposition scheme, exhibited superior performance than the standard COMPSO in multimodal problems.
(b) The proposed PCOMPSO model can be highly beneficial when the expected time-delay per function evaluation is relatively high.
(c) The total number of slave processes can have impact on the performance of PCOMPSO. If this number exceeds a relatively low threshold then the communication overhead increases significantly. This additional load was shown to reduce PCOMPSO's performance with respect to the required wall-clock time. In the experiments, it was shown that using up to 16 slave processes can produce satisfactory results for low time-delay level and dimension up to 1200.
(d) In complex problems, the time efficiency of PCOMPSO can increase with the problem's dimension when a relatively small number of slave processes is used.

Naturally, academic facilities typically serve a large number of users and they are characterized by heavy workload and communication overhead among the processors. Thus, it shall be underlined that the presented results are subject to mild variations depending on the status of the corresponding machine at the time of experimentation. This remark will become clearer in the next section, where the experimental results for a multicore desktop machine are presented.

5.2. Results for the desktop multicore system

The PCOMPSO approach was tested also on the desktop multicore system described in Section 4.1. In the experiments, 1, 5, and 8 CPUs were used. In the latter case, all the available processing threads of the system were occupied, while the typical processes of the operating system were concurrently running. This setting aimed at probing the time-performance of the algorithm under excessive system load.

Indeed, the obtained results revealed an interesting feature of multicore systems. More specifically, it was observed that, when a set of 30 experiments was assigned to 8 CPUs, the system did not provide all its processing threads immediately to the algorithm. Instead, it initially preserved one or more threads for performing operating system procedures, allocating the rest to the algorithm. Thus, for some of the initial experiments, some of the slave processes were running on same threads. As a consequence, higher wall-clock time was recorded for these experiments. After a few experiments, the system let the algorithm share all its processing threads. Thus, latter experiments required less time. These fluctuations of the required time per experiment are illustrated in Fig. 6 for a typical set of 30 experiments.

The rest of the experimental setting for the multicore system was identical to the academic cluster. All results are reported in Tables 11–15 and the time-performance is illustrated in Figs. 7–11, following the presentation motif of the previous section.

The results for the unimodal problem TP0 are similar to the corresponding results for the academic cluster. For the linear decomposition scheme, there are mostly statistically insignificant differences between the algorithms with respect to solution quality as reported in Table 11. On the other hand, the random

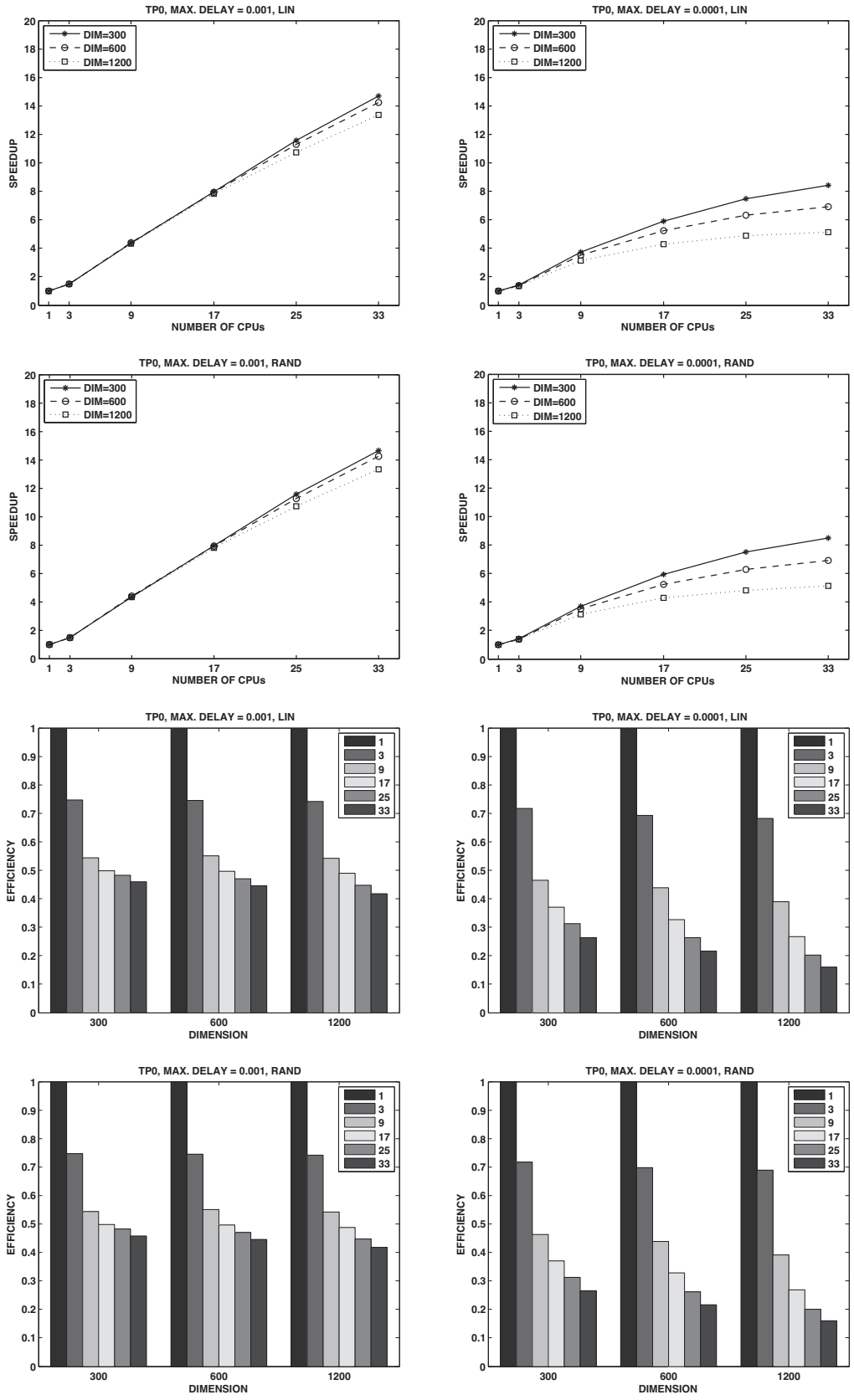

Fig. 1. Graphical illustrations for TP0 on the academic cluster.

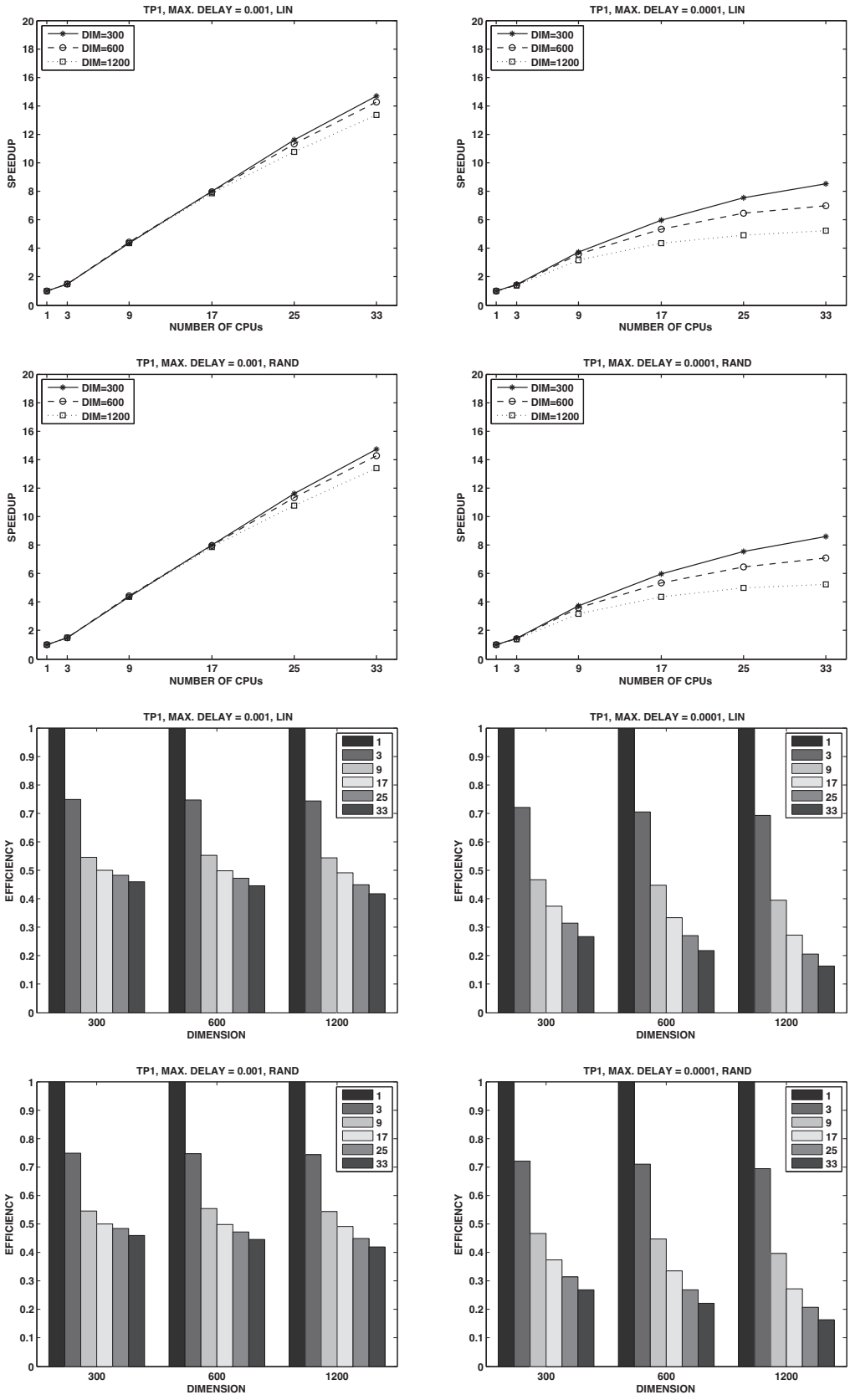

Fig. 2. Graphical illustrations for TP1 on the academic cluster.

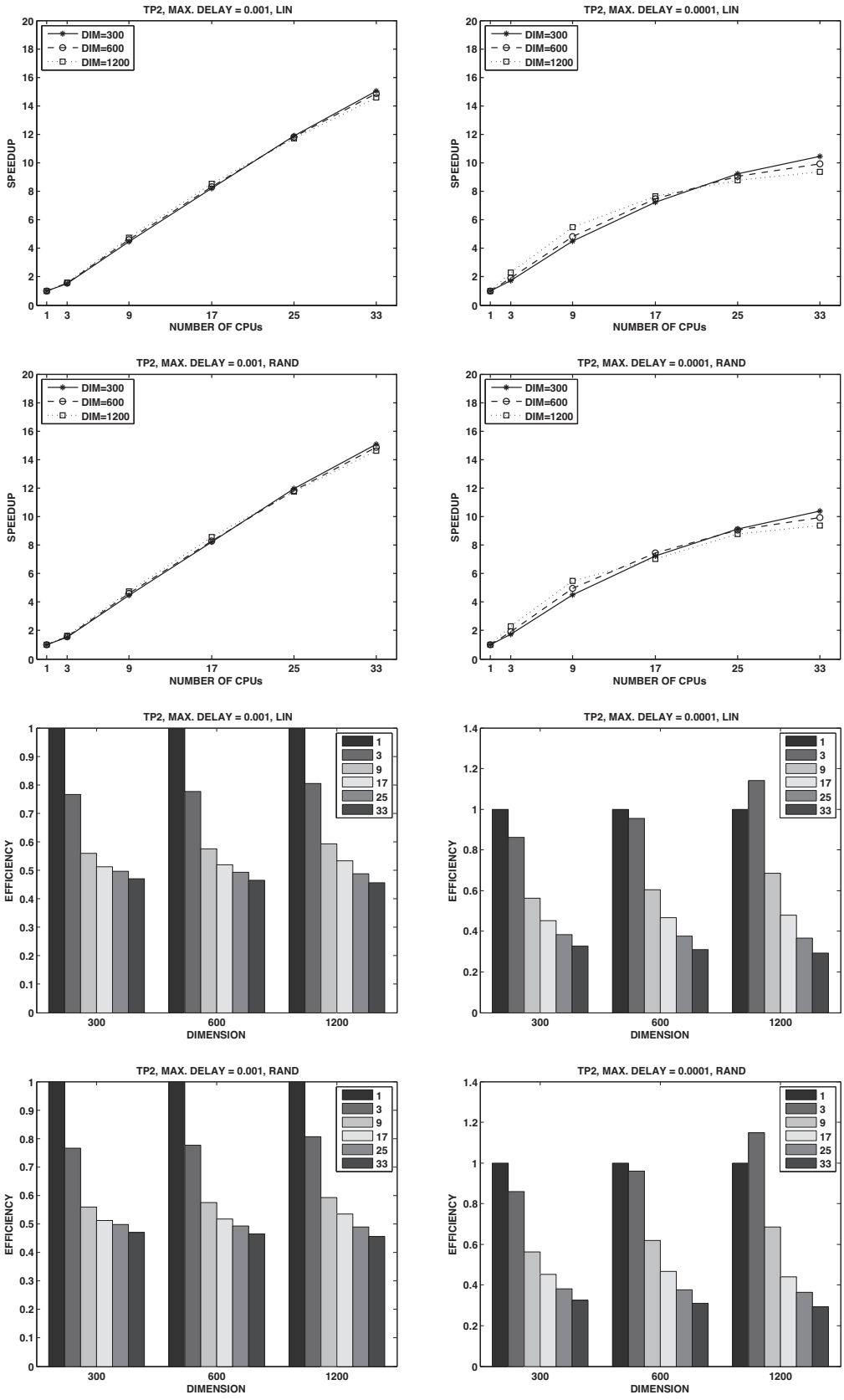

Fig. 3. Graphical illustrations for TP2 on the academic cluster.

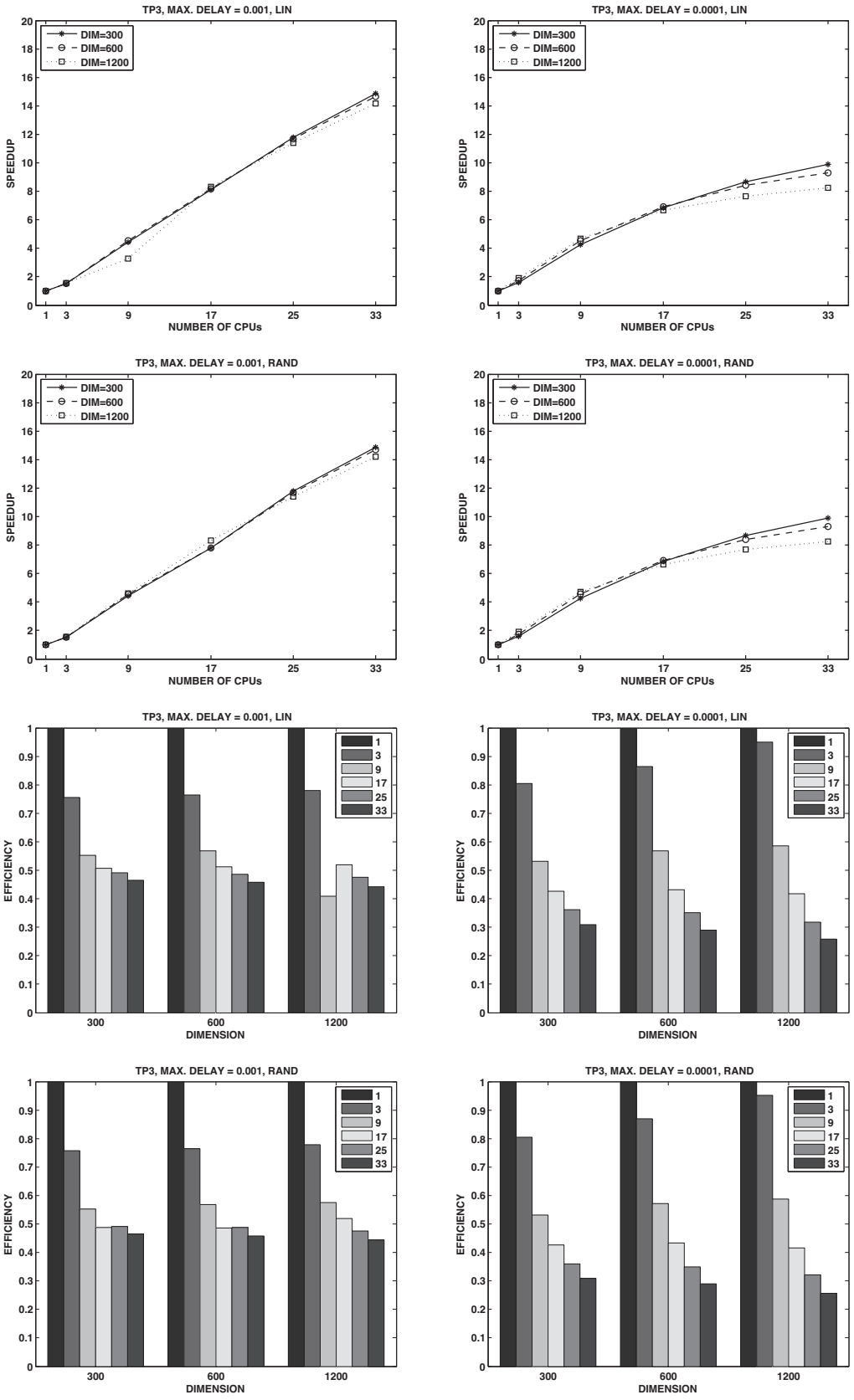

Fig. 4. Graphical illustrations for TP3 on the academic cluster.

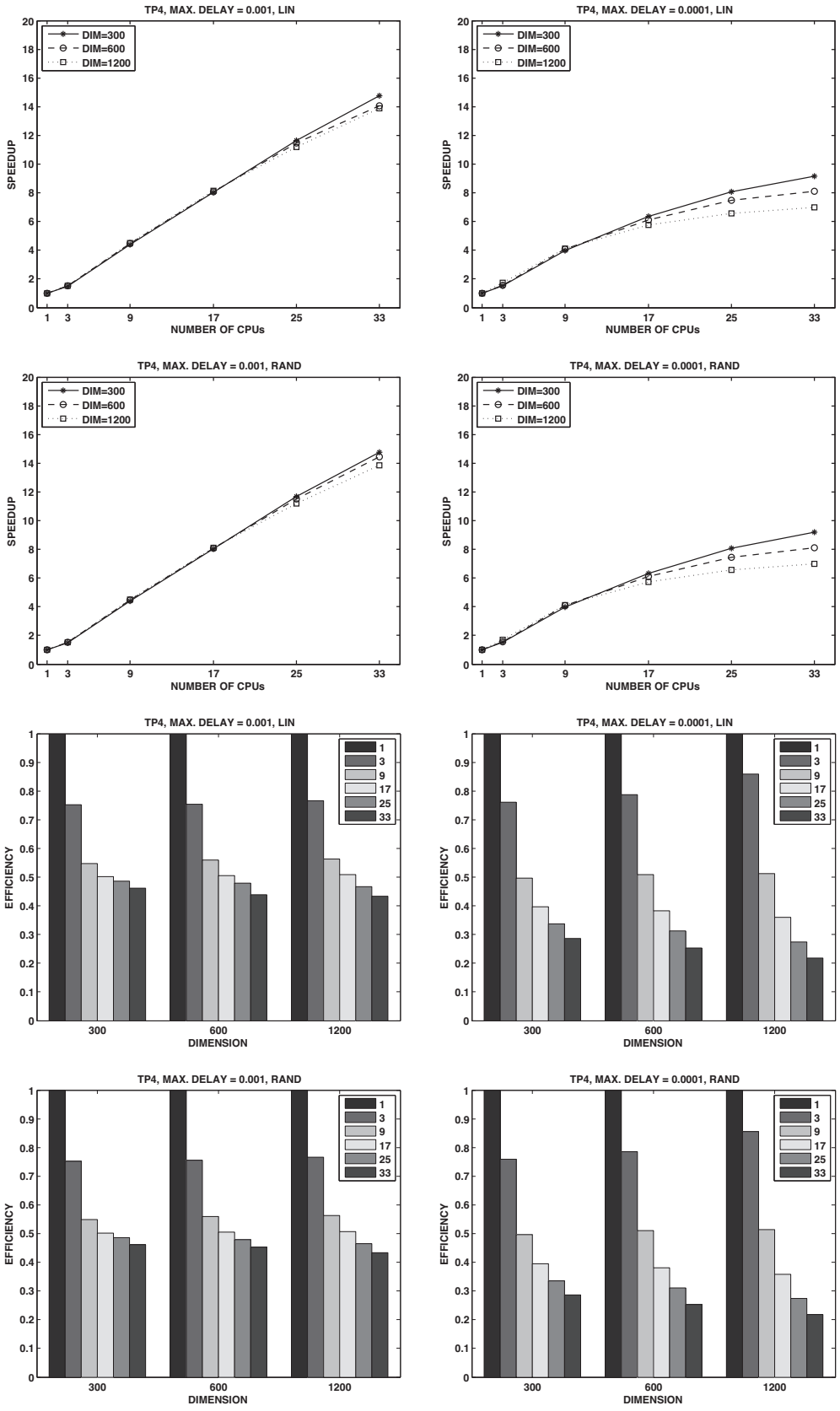

Fig. 5. Graphical illustrations for TP4 on the academic cluster.

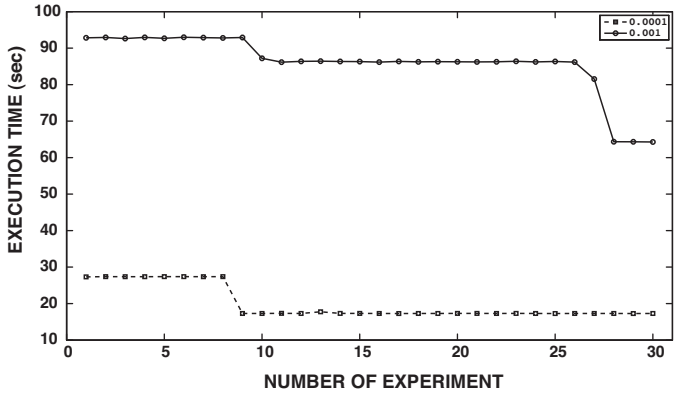

Fig. 6. The required wall-clock time per experiment in the desktop multicore system for the two time-delay levels.

decomposition scheme offered a significant advantage to the PCOMPSO approaches, which achieved considerably better solutions than the standard COMPSO regardless of the problem's dimension. Also, we can observe in Table 11 that the case of 5 CPUs habitually exhibits the most promising results.

The most interesting part of the analysis for the desktop system pertains to the required wall-clock time. In contrast to the results illustrated in Fig. 1 for TP0 on the academic cluster, Fig. 7 reveals that higher-dimensional instances of the problem are related to better or competitive speedup values as the number of CPUs increases. Also, the time efficiency for the 1200-dimensional problem instance is increasingly improved when 8 CPUs are used, while it deteriorates for 5 CPUs.

Table 11
Results for TP0 on the desktop multicore system.

Dim.		Number of CPUs		
		1 (COMPSO)	5	8
300	μ_f	**8.93e−26**	1.70e−24	7.35e−25
(Lin)	σ_f	**2.44e−25**	6.68e−24	2.57e−24
	$d_{\max_1}$	26.80 (0.20)	10.78 (0.01)	20.00 (4.52)
	$d_{\max_2}$	252.61 (0.37)	100.78 (0.15)	85.93 (8.00)
	cssd	None	None	None
	μ_f	8.93e−26	**7.08e−30**	9.78e−30
(Rand)	σ_f	2.44e−25	**5.34e−30**	7.30e−30
	$d_{\max_1}$	26.80 (0.20)	10.78 (0.01)	14.14 (0.10)
	$d_{\max_2}$	252.61 (0.37)	100.53 (0.23)	71.10 (8.54)
	cssd	All	All	All
600	μ_f	9.52e−22	**3.61e−23**	1.77e−22
(Lin)	σ_f	4.88e−21	**3.99e−23**	4.88e−22
	$d_{\max_1}$	54.26 (0.04)	23.70 (0.44)	24.11 (0.55)
	$d_{\max_2}$	506.63 (0.30)	203.20 (0.94)	149.64 (16.83)
	cssd	None	None	None
	μ_f	9.52e−22	**1.83e−26**	2.55e−26
(Rand)	σ_f	4.88e−21	**8.37e−27**	1.44e−26
	$d_{\max_1}$	54.26 (0.04)	22.44 (0.01)	23.79 (0.02)
	$d_{\max_2}$	506.63 (0.30)	201.95 (0.55)	146.34 (18.93)
	cssd	All	All	All
1200	μ_f	1.44e−19	**3.43e−20**	1.08e−18
(Lin)	σ_f	7.16e−19	**7.59e−20**	4.20e−18
	$d_{\max_1}$	116.89 (0.38)	53.17 (0.23)	37.59 (3.43)
	$d_{\max_2}$	1018.74 (2.04)	410.53 (2.14)	278.79 (28.10)
	cssd	8	None	1
	μ_f	1.44e−19	**3.11e−24**	5.22e−24
(Rand)	σ_f	7.16e−19	**9.01e−25**	2.17e−24
	$d_{\max_1}$	116.89 (0.38)	53.34 (0.07)	52.57 (0.03)
	$d_{\max_2}$	1018.74 (2.04)	406.97 (0.12)	284.24 (38.64)
	cssd	All	All	All

This performance deviation of PCOMPSO between the academic cluster and the desktop system can be attributed to their different architecture. The desktop system is a shared memory system, i.e., all slave processes have access to the same memory with the master process. Thus, there is no time-delay related to communication among different nodes for the transmission of data. On the other hand, in the case of the academic cluster the master process shall transmit all data mostly to different nodes (slave processes) through one or more hardware switches. Obviously, the latter procedure requires additional time that increases with the volume of the transmitted data. This is the main reason for the aforementioned performance differences. Additionally, it shall be highlighted that the observed performance trends appear to be independent of time-delay level and decomposition scheme.

Similar observations can be stated for test problem TP1, whose results are reported in Table 12 and illustrated in Fig. 8. Almost the same solution quality with the case of the academic cluster is observed in most experiments. Also, the time efficiency of the parallel approaches with 8 CPUs has an increasing trend, although with more fluctuations than in TP0. These fluctuations can be attributed to the specific time instance of the algorithm's execution, i.e., they are caused by the operating system. Also, there is no apparent relation of this effect with the decomposition scheme or the time-delay level.

The same pattern is repeated for the rest of the test problems. Fluctuations appear in the time efficiency of PCOMPSO as the number of CPUs reaches its maximal value. However, there is a clear trend of improvement with respect to the time-performance of PCOMPSO when a higher number of CPUs is used in higher problem dimensions, contrary to the corresponding behavior of the algorithm on the academic cluster.

Summarizing the experience gained from the application of the master–slave model of PCOMPSO on the desktop multicore system, we can state the following remarks:

(a) The performance of PCOMPSO is similar to that for the academic cluster with respect to solution quality.
(b) There is a significant wall-clock time gain when the proposed PCOMPSO model is used, especially under relatively high time-delay levels.
(c) The time-performance of PCOMPSO marginally differs than that for the academic cluster when a small number of CPUs is used.
(d) In contrast to the academic cluster, the time-performance of the algorithm can be improved in higher-dimensional problem instances when all the available CPUs are used. This improvement can be attributed to the different memory access and data transmission procedures between the two systems. However, in the case of the desktop system it is most likely that time-performance is affected by the operating system procedures when all the available CPU resources are used.

As a final remark, it can be stated that PCOMPSO exhibited notable consistency when less than the maximum number of CPUs of the multicore system were used. Thus, such systems can be considered as very useful and accessible hardware platforms for experimentation when a modest number of slave processes is required.

5.3. Further experimentation

The experimental results presented in the previous sections are based on a standard established test suite. However, in the past few years new sets of test problems have been developed to fulfill the necessity for more thorough investigations on problems of special type. Dynamic, multi-objective and constrained optimization are some of the research fields that have welcome such new challenging test suites. The new test problems have been

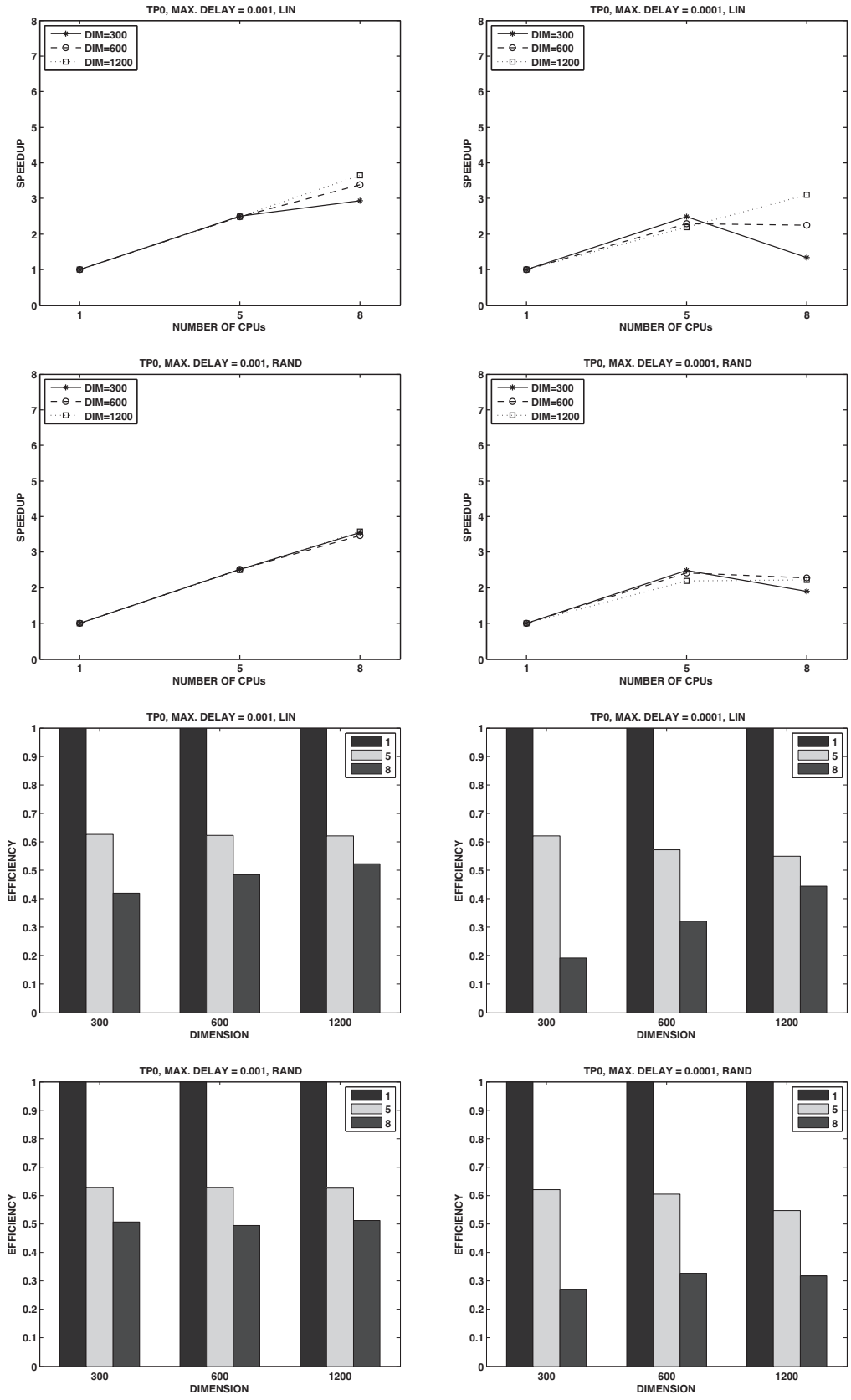

Fig. 7. Graphical illustrations for TP0 on the desktop multicore system.

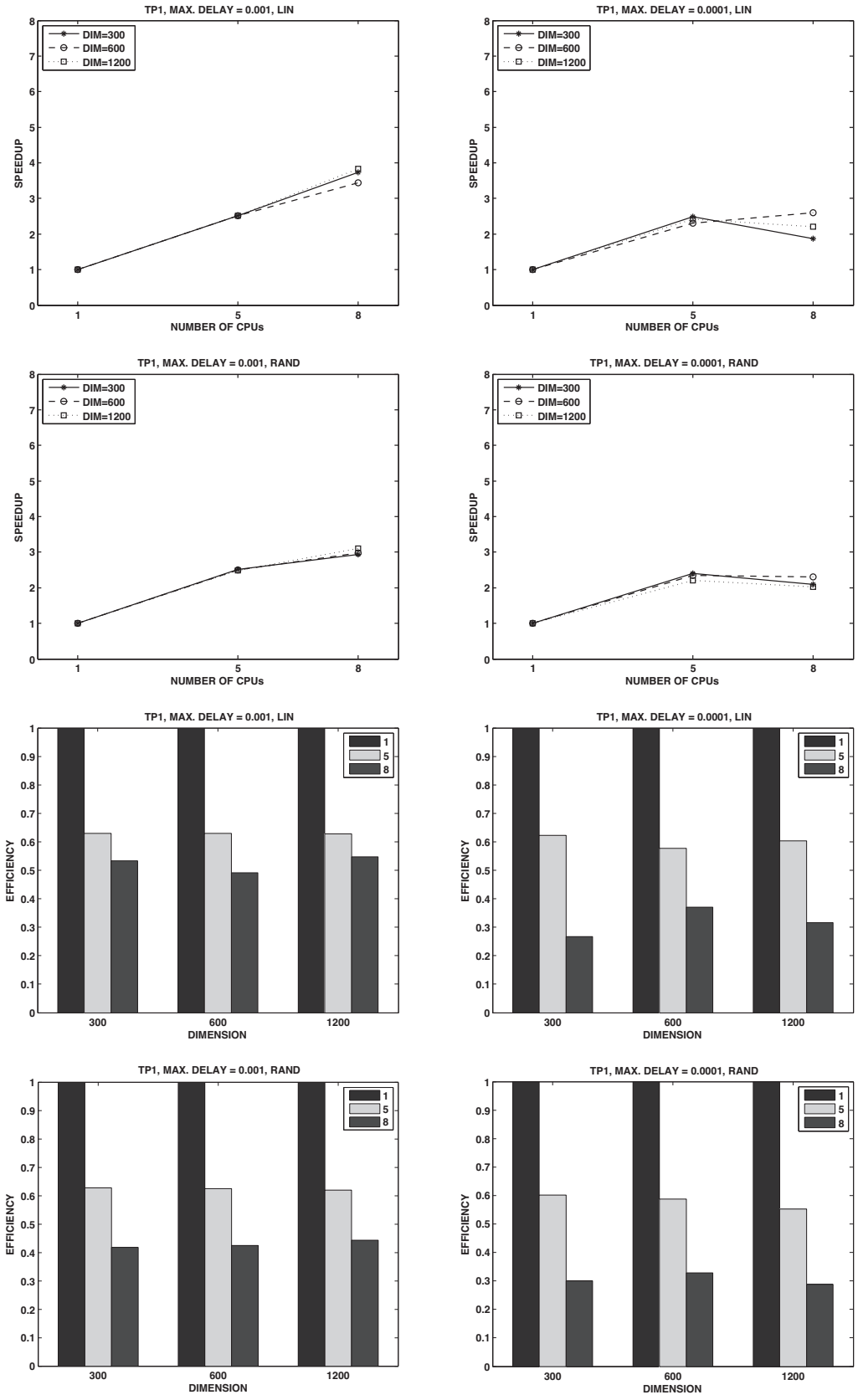

Fig. 8. Graphical illustrations for TP1 on the desktop multicore system.

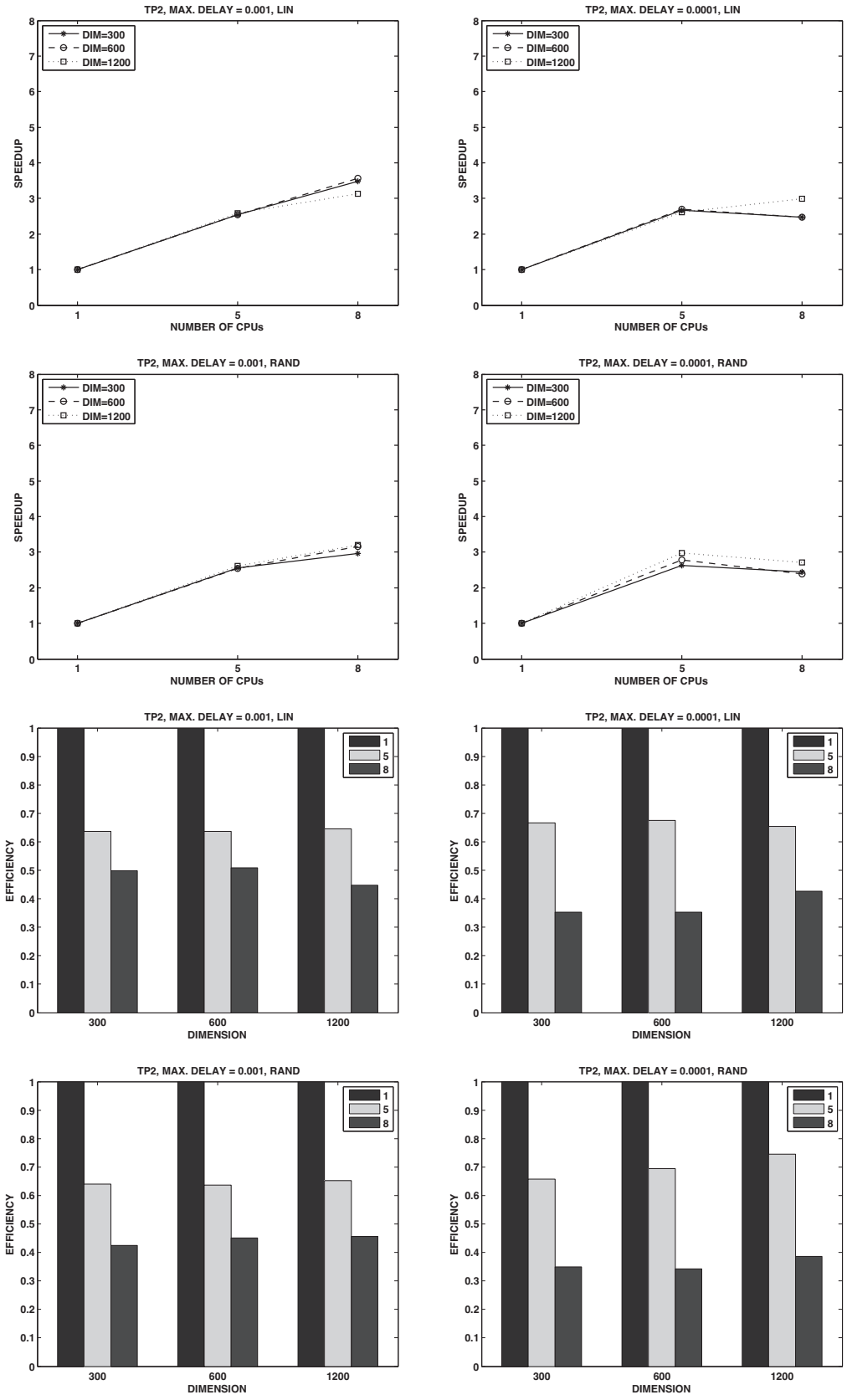

Fig. 9. Graphical illustrations for TP2 on the desktop multicore system.

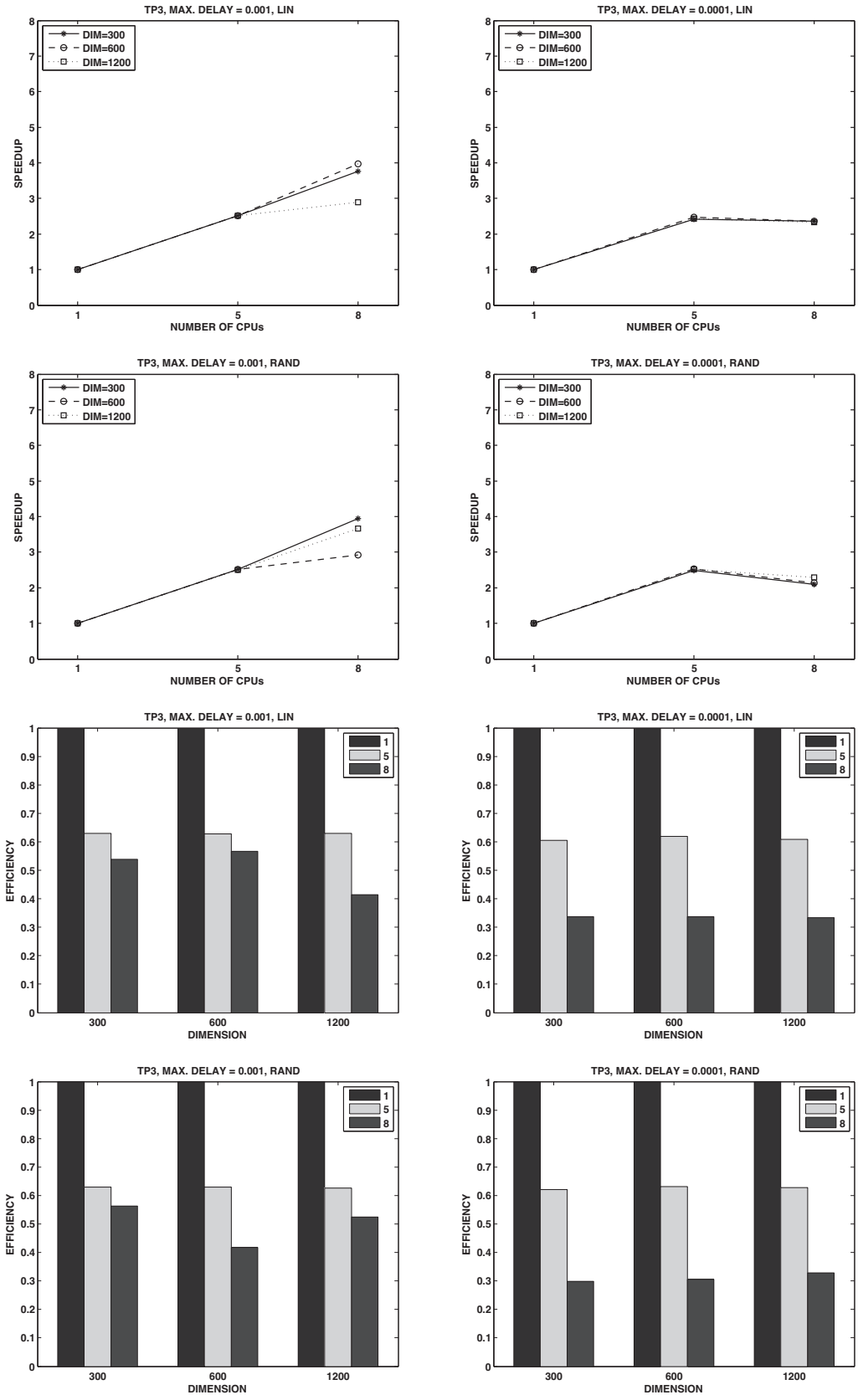

Fig. 10. Graphical illustrations for TP3 on the desktop multicore system.

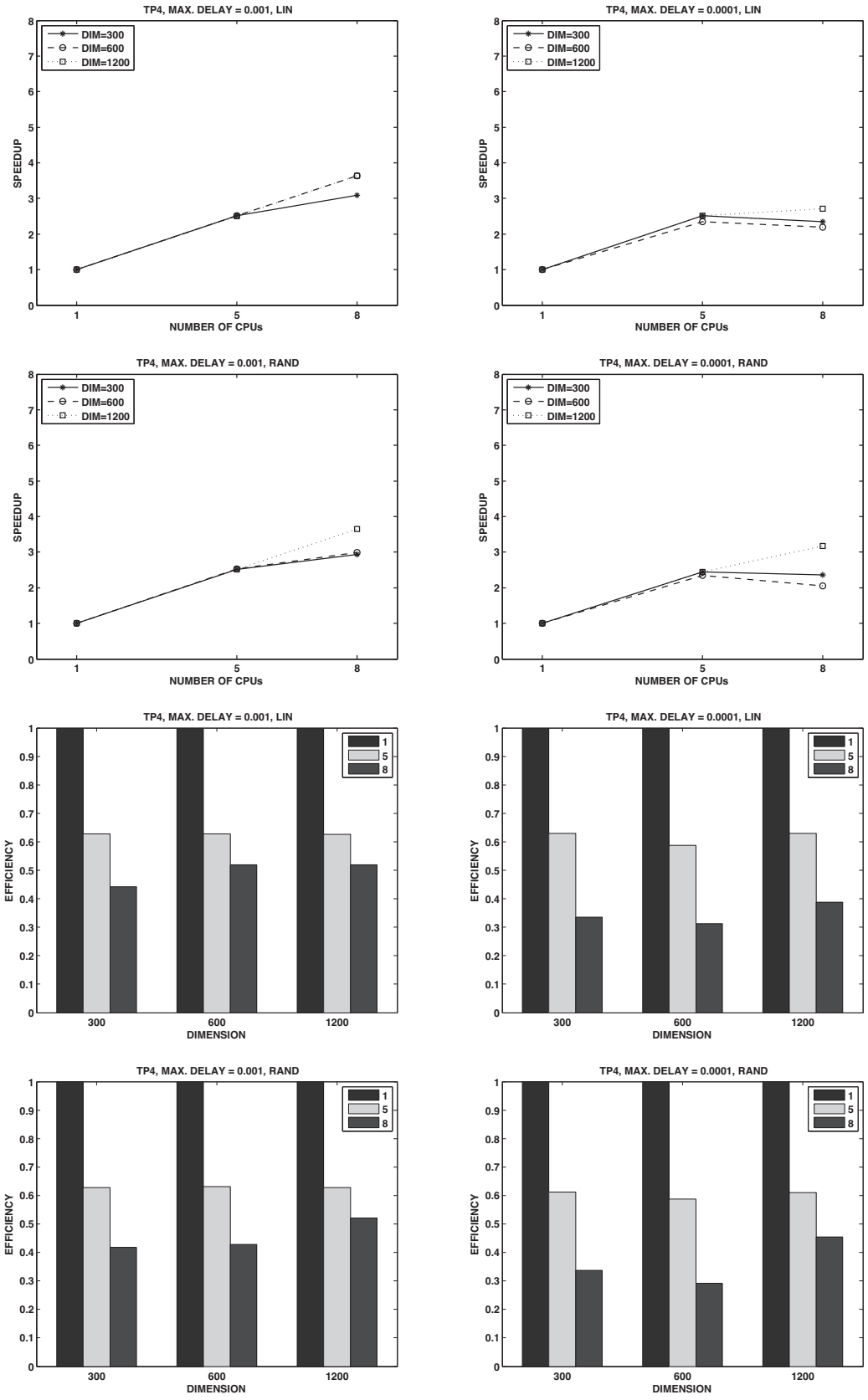

Fig. 11. Graphical illustrations for TP4 on the desktop multicore system.

Table 12
Results for TP1 on the desktop multicore system.

Dim.		Number of CPUs		
		1 (COMPSO)	5	8
300	μ_f	**3.51e+02**	3.68e+02	3.93e+02
(Lin)	σ_f	**5.83e+01**	5.10e+01	1.50e+02
	$d_{\max_1}$	27.04 (0.10)	10.86 (0.01)	14.44 (0.09)
	$d_{\max_2}$	253.18 (0.47)	100.67 (0.20)	67.74 (7.93)
	cssd	None	None	None
	μ_f	**3.51e+02**	1.13e+03	9.60e+02
(Rand)	σ_f	**5.83e+01**	9.79e+02	5.56e+02
	$d_{\max_1}$	27.04 (0.10)	11.25 (0.01)	12.88 (2.12)
	$d_{\max_2}$	253.18 (0.47)	100.77 (0.18)	86.44 (0.42)
	cssd	5, 8	1	1
600	μ_f	**7.34e+02**	7.35e+02	7.57e+02
(Lin)	σ_f	**6.38e+01**	6.75e+01	7.21e+01
	$d_{\max_1}$	55.61 (0.26)	24.09 (0.14)	21.42 (0.05)
	$d_{\max_2}$	508.37 (1.70)	202.08 (0.48)	147.93 (16.68)
	cssd	None	None	None
	μ_f	**7.34e+02**	1.74e+03	1.81e+03
(Rand)	σ_f	**6.38e+01**	7.32e+02	9.43e+02
	$d_{\max_1}$	55.61 (0.26)	23.66 (0.56)	24.19 (0.02)
	$d_{\max_2}$	508.37 (1.70)	203.19 (0.09)	170.76 (10.27)
	cssd	5, 8	1	1
1200	μ_f	1.52e+03	**1.50e+03**	1.50e+03
(Lin)	σ_f	1.06e+02	**1.04e+02**	1.07e+02
	$d_{\max_1}$	119.48 (0.87)	49.47 (0.13)	54.08 (0.71)
	$d_{\max_2}$	1023.38 (1.18)	408.02 (0.14)	266.83 (15.89)
	cssd	None	None	None
	μ_f	**1.52e+03**	2.99e+03	2.97e+03
(Rand)	σ_f	**1.06e+02**	6.17e+02	3.48e+02
	$d_{\max_1}$	119.48 (0.87)	53.99 (0.04)	59.15 (0.02)
	$d_{\max_2}$	1023.38 (1.18)	412.49 (1.40)	329.50 (0.23)
	cssd	5, 8	1	1

Table 13
Results for TP2 on the desktop multicore system.

Dim.		Number of CPUs		
		1 (COMPSO)	5	8
300	μ_f	**3.38e+02**	3.39e+02	3.48e+02
(Lin)	σ_f	**2.85e+01**	3.01e+01	3.58e+01
	$d_{\max_1}$	33.29 (0.30)	12.49 (0.03)	13.49 (0.03)
	$d_{\max_2}$	260.41 (4.92)	102.17 (0.09)	74.73 (10.97)
	cssd	None	None	None
	μ_f	3.38e+02	1.48e+01	**1.46e+01**
(Rand)	σ_f	2.85e+01	9.25e+00	**7.02e+00**
	$d_{\max_1}$	33.29 (0.30)	12.66 (0.06)	13.63 (0.68)
	$d_{\max_2}$	260.41 (4.92)	101.71 (0.08)	87.75 (0.18)
	cssd	5, 8	1	1
600	μ_f	6.59e+02	**6.54e+02**	6.58e+02
(Lin)	σ_f	3.65e+01	**3.72e+01**	5.69e+01
	$d_{\max_1}$	79.67 (0.40)	29.48 (0.17)	32.28 (0.09)
	$d_{\max_2}$	532.07 (0.51)	209.20 (0.14)	149.49 (22.63)
	cssd	None	None	None
	μ_f	6.59e+02	**3.34e+01**	3.61e+01
(Rand)	σ_f	3.65e+01	**1.09e+01**	1.16e+01
	$d_{\max_1}$	79.67 (0.40)	28.66 (0.10)	33.29 (0.21)
	$d_{\max_2}$	532.07 (0.51)	208.89 (0.15)	168.55 (0.22)
	cssd	5, 8	1	1
1200	μ_f	1.25e+03	**1.24e+03**	1.26e+03
(Lin)	σ_f	4.04e+01	**4.00e+01**	6.97e+01
	$d_{\max_1}$	219.09 (0.95)	83.72 (0.32)	73.36 (1.36)
	$d_{\max_2}$	1124.17 (1.69)	436.00 (1.19)	359.09 (6.25)
	cssd	None	None	None
	μ_f	1.25e+03	**7.11e+01**	7.86e+01
(Rand)	σ_f	4.04e+01	**2.05e+01**	2.24e+01
	$d_{\max_1}$	219.09 (0.95)	73.47 (1.04)	80.92 (0.36)
	$d_{\max_2}$	1124.17 (1.69)	430.41 (1.60)	351.39 (0.52)
	cssd	5, 8	1	1

either developed from scratch or derived through proper mathematical manipulations of the existing test problems.

In the field of large-scale optimization, there has been intense research activity towards the production of new test suites. A remarkable effort has been paid by researchers at the Nature Inspired Computation and Applications Laboratory at the University of Science and Technology of China[4]. The outcome of this effort was a test suite for large-scale optimization, which served as the main benchmarking and competition ground at the corresponding special session of the IEEE 2008 Congress on Evolutionary Computation (IEEE CEC'08) [50].

The test suite consists of unimodal, multimodal, separable and non-separable problems. More specifically, it consists of modified versions of the test problems TP0–TP4 that were considered in the previous sections. In their new versions, the test problems are rotated and shifted producing landscapes that constitute hard challenges for nature-inspired optimization algorithms. Besides the aforementioned five test problems, two additional problems were also included in the test suite, namely Schwefel and FastFractal "DoubleDip", with all problems being fully scalable for dimension up to 1000.

This test suite was used for further experimentation with PCOMPSO, in order to assess its capability on tackling harder, large-scale optimization problems than the ones reported in the previous sections. In this set of experiments, attention was focused on the PCOMPSO's efficiency in terms of the obtained solution quality. Thus, its time-performance was neglected although the same parallel version as in the previous sections was used. The experiments were conducted on the same multicore machine considered in Section 5.2, using the test suite's C++source code[5].

The developers of the test suite denote the test problems as follows [50]: F1 for the Shifted Sphere problem; F2 for Shifted Schwefel; F3 for Shifted Rosenbrock; F4 for Shifted Rastrigin; F5 for Shifted Griewank; and F6 for Shifted Ackley. This notation is adopted in the present analysis[6]. The complete definitions of the test problems are omitted due to space limitations and the reader is referred to the original source [50].

PCOMPSO was applied on the 1000-dimensional instances of the test problems F1–F6, using the experimental configuration that was used for the IEEE CEC'08 competition [50]. More specifically, 25 independent experiments were conducted for each test problem, allowing the algorithm to perform a total of 5×10^6 function evaluations. PCOMPSO's most promising scheme, i.e., the one with the random decomposition, was used with 500 subswarms, each one consisting of five 2-dimensional particles. It shall be reminded that the subswarms must have small sizes and, hence, assume low-dimensional particles to avoid efficiency issues as discussed in Section 2.2.

The average performance of PCOMPSO was compared to that of two PSO-based and one DE-based approaches specialized on large-scale optimization problems, which contested in the IEEE CEC'08 competition. These approaches are the *efficient population utilization strategy for particle swarm optimizer* (EPUS-PSO) [51], the *dynamic multi-swarm particle swarm optimizer with local search*

[4] http://nical.ustc.edu.cn/

[5] http://nical.ustc.edu.cn/conferences/cec2008/lsgo/LSGO.CEC08.Benchmark.zip

[6] The compilation of the FastFractal "DoubleDip" test problem was not stable in the experimentation machine due to its Java modules, therefore experiments were omitted for this problem.

Table 14
Results for TP3 on the desktop multicore system.

Dim.		Number of CPUs		
		1 (COMPSO)	5	8
300	μ_f	6.32e−02	4.28e−02	**1.06e−01**
(Lin)	σ_f	1.31e−01	6.55e−02	**3.13e−01**
	$d_{\max_1}$	31.92 (0.12)	13.19 (0.25)	13.51 (0.06)
	$d_{\max_2}$	257.78 (0.32)	102.44 (0.15)	68.42 (7.76)
	cssd	None	None	None
	μ_f	6.32e−02	**1.11e−02**	6.74e−03
(Rand)	σ_f	1.31e−01	**2.65e−02**	1.74e−02
	$d_{\max_1}$	31.92 (0.12)	12.87 (0.16)	15.23 (0.02)
	$d_{\max_2}$	257.78 (0.32)	102.44 (0.13)	65.35 (0.04)
	cssd	5, 8	1	1
600	μ_f	**6.61e−02**	6.33e−02	7.74e−02
(Lin)	σ_f	**1.45e−01**	1.25e−01	1.87e−01
	$d_{\max_1}$	75.73 (0.33)	30.58 (0.26)	32.02 (0.13)
	$d_{\max_2}$	527.01 (0.63)	209.60 (0.39)	132.80 (0.09)
	cssd	None	None	None
	μ_f	**6.61e−02**	4.94e−03	6.57e−03
(Rand)	σ_f	**1.45e−01**	1.15e−02	1.24e−02
	$d_{\max_1}$	75.73 (0.33)	29.97 (0.12)	35.39 (0.07)
	$d_{\max_2}$	527.01 (0.63)	209.16 (0.11)	180.43 (0.18)
	cssd	5, 8	1	1
1200	μ_f	**4.06e−02**	9.96e−02	4.62e−02
(Lin)	σ_f	**5.97e−02**	2.46e−01	6.02e−02
	$d_{\max_1}$	197.40 (0.84)	81.07 (1.37)	84.66 (0.31)
	$d_{\max_2}$	1100.65 (1.13)	437.02 (0.79)	380.38 (0.32)
	cssd	None	None	None
	μ_f	**4.06e−02**	6.39e−03	6.03e−03
(Rand)	σ_f	**5.97e−02**	9.77e−03	1.06e−02
	$d_{\max_1}$	197.40 (0.84)	78.67 (0.17)	85.95 (0.20)
	$d_{\max_2}$	1100.65 (1.13)	439.05 (1.15)	300.14 (39.87)
	cssd	5, 8	1	1

Table 15
Results for TP4 on the desktop multicore system.

Dim.		Number of CPUs		
		1 (COMPSO)	5	8
300	μ_f	**4.00e−13**	4.38e−13	4.82e−13
(Lin)	σ_f	**4.65e−14**	1.99e−13	4.18e−13
	$d_{\max_1}$	29.35 (1.18)	11.66 (0.02)	12.47 (0.03)
	$d_{\max_2}$	255.33 (0.49)	101.60 (0.06)	82.65 (0.13)
	cssd	None	None	None
	μ_f	4.00e−13	2.69e−13	**2.63e−13**
(Rand)	σ_f	4.65e−14	2.18e−14	**2.20e−14**
	$d_{\max_1}$	29.35 (1.18)	11.98 (0.20)	12.41 (0.01)
	$d_{\max_2}$	255.33 (0.49)	101.55 (0.06)	87.18 (0.11)
	cssd	5, 8	1	1
600	μ_f	4.26e−02	**1.19e−12**	1.21e−12
(Lin)	σ_f	2.33e−01	**1.28e−12**	7.63e−13
	$d_{\max_1}$	65.39 (1.14)	27.82 (0.15)	29.84 (0.04)
	$d_{\max_2}$	517.79 (1.11)	206.17 (0.48)	142.64 (19.75)
	cssd	None	None	None
	μ_f	4.26e−02	**5.63e−13**	5.65e−13
(Rand)	σ_f	2.33e−01	**3.31e−14**	3.51e−14
	$d_{\max_1}$	65.39 (1.14)	27.84 (0.03)	31.95 (0.03)
	$d_{\max_2}$	517.79 (1.11)	205.31 (0.11)	173.07 (12.63)
	cssd	5, 8	1	1
1200	μ_f	1.15e−11	1.44e−11	**9.58e−12**
(Lin)	σ_f	1.58e−11	2.75e−11	**5.30e−12**
	$d_{\max_1}$	163.84 (3.55)	65.08 (0.22)	60.35 (4.54)
	$d_{\max_2}$	1068.61 (4.32)	426.18 (0.65)	293.90 (39.65)
	cssd	None	None	None
	μ_f	1.15e−11	**1.22e−12**	1.23e−12
(Rand)	σ_f	1.58e−11	**4.70e−14**	3.84e−14
	$d_{\max_1}$	163.84 (3.55)	67.18 (0.17)	51.55 (7.86)
	$d_{\max_2}$	1068.61 (4.32)	425.18 (0.81)	293.20 (39.40)
	cssd	5, 8	1	1

Table 16
Average performance of PCOMPSO, EPUS-PSO, DMS-PSO and DEwSAcc on the 1000-dimensional instance of the test suite used in the IEEE CEC'08 competition on large scale global optimization.

Algorithm	Stat.	F1	F2	F3	F4	F5	F6
PCOMPSO	Mean	1.76e+03	4.66e+01	2.46e+07	7.59e+02	1.66e+01	2.45e+00
	St.D.	6.05e+02	1.15e+00	1.82e+07	3.60e+01	4.91e+00	2.01e−01
EPUS-PSO	Mean	5.53e+02	4.66e+01	8.37e+05	7.58e+03	5.89e+00	1.89e+01
	St.D.	2.86e+01	4.00e−01	1.52e+05	1.51e+02	3.91e−01	2.49e+00
DMS-PSO	Mean	0.00e+00*	9.15e+01	8.98e+09	3.84e+03	0.00e+00*	7.76e+00
	St.D.	0.00e+00*	7.14e−01	4.39e+08	1.71e+02	0.00e+00*	8.92e−02
DEwSAcc	Mean	8.79e−03	9.61e+01	9.15e+03	1.82e+03	3.58e−03	2.30e+00
	St.D.	5.27e−03	1.82e+00	1.26e+03	1.38e+02	5.74e−03	2.98e−01

(DMS-PSO) [52], and the *differential evolution with self-adaptation and cooperative co-evolution* (DEwSAcc) [53].

The comparisons with respect to the mean value and standard deviation of the obtained solution errors are reported in Table 16. The values reported for EPUS-PSO, DMS-PSO and DEwSAcc were adopted directly from the original sources. The results of DMS-PSO marked with an asterisk (*) in Table 16 appear as pure zeros in the original source (the lowest non-zero value in that table was of order 10^{-2}). As we see in the table, there is no algorithm that dominates all others in all problems. Excluding the convex unimodal problem F1, PCOMPSO was able to produce competitive results with the rest of the algorithms. In fact, for F2, F4 and F6, it was the best performing PSO-based approach.

This is remarkable, keeping in mind that in contrast to other approaches PCOMPSO does not employ complex strategies or local search to fine-tune the obtained solutions. Obviously, the lack of such a mechanism is highly responsible for its inability to move as close to the global minimizer of F1 as the rest of the algorithms. Nevertheless, its performance suggests that, providing additional search mechanisms such as local search or adaptive parameter tuning, PCOMPSO can become a very competitive approach.

6. Conclusions

An essential master–slave model for the parallelization of the COMPSO algorithm was introduced. The proposed PCOMPSO approach was implemented using the MPI standard. Its performance was assessed on a set of five widely used test problems where additional random time–delays were considered in the function evaluations. Besides that, a random decomposition scheme was incorporated to the algorithm to enhance its performance with respect to solution quality.

The proposed approach was implemented on two different computer systems. The first one was an academic cluster, namely the Saw facility of the SHARCNET grid network. The second one was a shared-memory multicore PC based on an Intel® I7 processor. The results were statistically analyzed with respect to the obtained solution quality and time-performance. The analysis revealed some significant aspects of the algorithm. First, it suggested that PCOMPSO can be a valuable tool, outperforming COMPSO in time-demanding problems. Second, it showed the superiority of the random decomposition scheme against the linear one in general problems. Finally, it offered intuition regarding the time-performance of PCOMPSO in different parallel hardware platforms.

The experimental results suggested that, despite the limited available resources, PCOMPSO implementations on shared-memory systems can offer satisfactory results in high-dimensional problem instances. This holds even when the demand for processing resources is pushed to the limit. However, it is accompanied by a possible sensitivity on the time instance of the algorithm's execution, i.e., on the operating system procedures running at the time of experimentation.

On the other hand, in large scale systems with hundreds of processors, the communication burden shall be seriously taken into consideration. The performance of the algorithm is related to the number of processors, which shall be kept in relatively small values. Using more CPUs does not guarantee performance improvement even in higher–dimensional problem instances.

Moreover, comparisons of PCOMPSO with other PSO-based and DE-based approaches on recently proposed large scale optimization test suites, suggest that it can be competitive to more sophisticated approaches that employ complex update rules and local search. This is a strong motivation for further experimentation with memetic variants of the algorithm.

Future research will also include the development of alternative parallel implementations of PCOMPSO with an emphasis on the *island model*. This model requires substantial modifications to the algorithm's structure and operation, as well as a decentralized communication system among the subswarms.

Acknowledgements

The author wishes to thank the anonymous reviewers for their valuable comments and suggestions.

Appendix A.

Test problem 0 (TP0 – Sphere) [15]. This is a separable n-dimensional problem, defined as:

$$f(x) = \sum_{i=1}^{n} x_i^2, \tag{9}$$

with $x \in [-100, 100]^n$, and it has one global minimizer, $x^* = (0, 0, \ldots, 0)^\top$, with $f(x^*) = 0$.

Test problem 1 (TP1 - Generalized Rosenbrock) [15]. This is a non-separable n-dimensional problem, defined as:

$$f(x) = \sum_{i=1}^{n-1} (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2), \tag{10}$$

with $x \in [-30, 30]^n$, and it has a global minimizer, $x^* = (1, 1, \ldots, 1)^\top$, with $f(x^*) = 0$.

Test problem 2 (TP2 – Rastrigin) [15]. This is a separable n-dimensional problem, defined as:

$$f(x) = 10n + \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i)), \tag{11}$$

with $x \in [-5.12, 5.12]^n$, and it has a global minimizer, $x^* = (0, 0, \ldots, 0)^\top$, with $f(x^*) = 0$.

Test problem 3 (TP3 – Griewank) [15]. This is a non–separable n-dimensional problem, defined as:

$$f(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1, \tag{12}$$

with $x \in [-600, 600]^n$, and it has a global minimizer, $x^* = (0, 0, \ldots, 0)^\top$, with $f(x^*) = 0$.

Test problem 4 (TP4 - Ackley) [15]. This is a non-separable n-dimensional problem, defined as:

$$f(x) = 20 + \exp(1) - 20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n} \cos(2\pi x_i)\right), \tag{13}$$

with $x \in [-20, 30]^n$, and it has a global minimizer, $x^* = (0, 0, \ldots, 0)^\top$, with $f(x^*) = 0$.

References

[1] M.R. AlRashidi, M.E. El-Hawary, A survey of particle swarm optimization applications in electric power systems, IEEE Transactions on Evolutionary Computation 13 (4) (2009) 913–918.
[2] J. Robinson, Y. Rahmat-Samii, Particle swarm optimization in electromagnetics, IEEE Transactions on Antennas and Propagation 52 (2004) 397–407.
[3] J. Park, K. Choi, D.J. Allstot, Parasitic-aware RF circuit design and optimization, IEEE Transactions on Circuits and Systems I: Regular Papers 51 (2004) 1953–1966.
[4] D.W. Boeringer, D.H. Werner, Particle swarm optimization versus genetic algorithms for phased array synthesis, IEEE Transactions on Antennas and Propagation 52 (3) (2004) 771–779.
[5] Y.G. Petalas, K.E. Parsopoulos, M.N. Vrahatis, Improving fuzzy cognitive maps learning through memetic particle swarm optimization, Soft Computing 13 (1) (2009) 77–94.
[6] B. Liu, L. Wang, Y.-H. Jin, An effective pso-based memetic algorithm for flow shop scheduling, IEEE Transactions on Systems, Man and Cybernetics, Part B 37 (2007) 18–27.
[7] K. Parsopoulos, F. Kariotou, G. Dassios, M. Vrahatis, Tackling magnetoencephalography with particle swarm optimization, International Journal of Bio-Inspired Computation 1 (1/2) (2009) 32–49.
[8] R. Xu, D.C. Anagnostopoulos, D.C. Wunsch, Multiclass cancer classification using semisupervised ellipsoid artmap and particle swarm optimization with gene expression data, IEEE/ACM Transactions on Computational Biology and Bioinformatics 4 (1) (2007) 65–77.
[9] C. Skokos, K.E. Parsopoulos, P.A. Patsis, M.N. Vrahatis, Particle swarm optimization: An efficient method for tracing periodic orbits in 3D galactic potentials, Monthly Notices of the Royal Astronomical Society 359 (2005) 251–260.
[10] A. Banks, J. Vincent, C. Anyakoha, A review of particle swarm optimization. Part I: background and development, Natural Computing 6 (4) (2007) 467–484.
[11] A. Banks, J. Vincent, C. Anyakoha, A review of particle swarm optimization. Part II: hybridisation combinatorial multicriteria and constrained optimization and indicative applications, Natural Computing 7 (1) (2008) 109–124.
[12] R. Poli, An Analysis of Publications on Particle Swarm Optimisation Applications, Tech. Rep. CSM-649, University of Essex, Department of Computer Science, UK, 2007.
[13] M. Clerc, Particle Swarm Optimization, ISTE Ltd, London, UK, 2006.
[14] A.P. Engelbrecht, Fundamentals of Computational Swarm Intelligence, Wiley, Chichester, UK, 2006.
[15] K.E. Parsopoulos, M.N. Vrahatis, Particle Swarm Optimization and Intelligence: Advances and Applications, Information Science Publishing (IGI Global), Hershey (PA), USA, 2010.
[16] W.B. Powell, Approximate Dynamic Programming: Solving the Curses of Dimensionality, Wiley, New York, USA, 2007.
[17] M.A. Potter, K. De Jong, Cooperative coevolution: An architecture for evolving coadapted subcomponents, Evolutionary Computation 8 (1) (2000) 1–29.
[18] G. Sánchez-Ante, F. Ramos, J. Frausto, Cooperative simulated annealing for path planning in multi-robot systems, in: Proceedings of the Mexican International Conference on Artificial Intelligence (MICAI 2000), Acapulco, Mexico, Vol. 1793 of Lecture Notes in Computer Science, Springer, Berlin, Germany, 2000, pp. 148–157.
[19] T.G. Crainic, M. Grendeau, Cooperative parallel tabu search for capacitated network design, Journal of Heuristics 8 (2002) 601–627.
[20] M.A. Potter, K. De Jong, A cooperative coevolutionary approach to function optimization, in: Y. Davidor, H.-P. Schwefel (Eds.), Proc. 3rd Conference on Parallel Problem Solving from Nature, Springer-Verlag, Berlin, Germany, 1994, pp. 249–257.
[21] K. Doerner, R.F. Hartl, M. Reimann, Cooperative ant colonies for optimizing resource allocation in transportation, in: Lecture Notes in Computer Science, Vol. 2037, Springer, Berlin, Germany, 2001, pp. 70–79.
[22] D. Sofge, K. De Jong, A. Schultz, A blended population approach to cooperative coevoultion for decomposition of complex problems, in: Proc. 2002 IEEE Congress on Evolutionary Computation (CEC'02), 2002, pp. 413–418.
[23] B. Niu, Y. Zhu, X. He, Multi-population cooperative particle swarm optimization, in: Proc. 2005 European Conference on Artificial Life, 2005, pp. 874–883.
[24] M. El-Abd, M.S. Kamel, A taxonomy of cooperative particle swarm optimizers, International Journal of Computational Intelligence Research 4 (2) (2008) 137–144.
[25] M. Köppen, K. Franke, R. Vicente-Garcia, Tiny GAs for image processing applications, IEEE Computational Intelligence Magazine 1 (2) (2006) 17–26.
[26] T. Huang, A.S. Mohan, Micro-particle swarm optimizer for solving high dimensional optimization problems μPSO for high dimensional optimization problems, Applied Mathematics and Computation 181 (2) (2006) 1148–1154.
[27] K.E. Parsopoulos, Cooperative micro-particle swarm optimization, in: ACM 2009 World Summit on Genetic and Evolutionary Computation (2009 GEC Summit), Shanghai, China, 2009, pp. 467–474.
[28] K.E. Parsopoulos, Cooperative micro-differential evolution for high-dimensional problems, in: Genetic and Evolutionary Computation Conference 2009 (GECCO'09), Montreal, Canada, 2009, pp. 531–538.
[29] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: Proc. IEEE Int. Conf. Neural Networks, Vol. IV, IEEE Service Center, Piscataway, NJ, 1995, pp. 1942–1948.
[30] J. Kennedy, R.C. Eberhart, Swarm Intelligence, Morgan Kaufmann Publishers, San Francisco (CA), USA, 2001.
[31] J. Kennedy, Small worlds and mega-minds: Effects of neighborhood topology on particle swarm performance, in: Proc. IEEE Congr. Evol. Comput., IEEE Press, Washington, DC, USA, 1999, pp. 1931–1938.
[32] M. Clerc, J. Kennedy, The particle swarm-explosion, stability, and convergence in a multidimensional complex space, IEEE Transactions on Evolutionary Computation 6 (1) (2002) 58–73.
[33] I.C. Trelea, The particle swarm optimization algorithm: convergence analysis and parameter selection, Information Processing Letters 85 (2003) 317–325.
[34] K.E. Parsopoulos, M.N. Vrahatis, On the computation of all global minimizers through particle swarm optimization, IEEE Transactions on Evolutionary Computation 8 (3) (2004) 211–224.
[35] M. El-Abd, Cooperative models of particle swarm optimizers, Ph.D. Thesis, Dept. Elect. Comput. Eng., Univ. Waterloo, Waterloo, Ontario, Canada, 2008.
[36] F. Van den Bergh, A.P. Engelbrecht, A cooperative approach to particle swarm optimization, IEEE Transactions on Evolutionary Computation 8 (3) (2004) 225–239.
[37] L. Baldo, L. Brenner, L.G. Fernandes, P. Fernandes, A. Sales, Performance models for master/slave parallel programs, Electronic Notes in Theoretical Computer Science 128 (4) (2005) 101–121.
[38] S. Sahni, G. Vairaktarakis, The master–slave paradigm in parallel computer and industrial settings, Journal of Global Optimization 9 (3-4) (2004) 357–377.
[39] E. Alba (Ed.), Parallel Metaheuristics, Wiley-Interscience, Hoboken, NJ, 2005.
[40] E. Cantú-Paz, Efficient and Accurate Parallel Genetic Algorithms, Kluwer Academic Publishers, Boston, MA, 2000.
[41] M. Belal, T. El-Ghazawi, Parallel models for particle swarm optimizers, International Journal on Intelligent Cooperative Information Systems 4 (1) (2004) 100–111.
[42] M. Waintraub, R. Schirru, C.M.N.A. Pereira, Multiprocessor modeling of parallel particle swarm optimization applied to nuclear engineering problems, Progress in Nuclear Energy 51 (6–7) (2009) 680–688.
[43] Y. Li, Y. Cao, Z. Liu, Y. Liu, Q. Jiang, Dynamic optimal reactive power dispatch based on parallel particle swarm optimization algorithm, Computers & Mathematics with Applications 57 (11-12) (2009) 1835–1842.
[44] Y. Zhang, D. Gallipoli, C.E. Augarde, Simulation-based calibration of geotechnical parameters using parallel hybrid moving boundary particle swarm optimization, Computers and Geotechnics 36 (4) (2009) 604–615.
[45] A.B. de Carvalho, A. Pozo, Mining rules: a parallel multiobjective particle swarm optimization approach, in: Swarm Intelligence for Multi-Objective Problems in Data Mining, Vol. 242 of Studies in Computational Intelligence, Springer, Berlin, Germany, 2009, pp. 179–198.
[46] A. Farmahini-Farahani, S. Vakili, S.M. Fakhraie, S. Safari, C. Lucas, Parallel scalable hardware implementation of asynchronous discrete particle swarm optimization, Engineering Applications of Artificial Intelligence 23 (2) (2010) 177–187.
[47] I.T. Jolliffe, Principal Component Analysis, Springer, Secaucus (NJ), USA, 2002.
[48] Z. Yang, K. Tang, X. Yao, Large scale evolutionary optimization using cooperative coevolution, Information Sciences 178 (15) (2008) 2986–2999.
[49] X. Li, X. Yao, Tackling high dimensional nonseparable optimization problems by cooperatively coevolving particle swarms, in: Proc. IEEE 2009 Congress on Evolutionary Computation (IEEE CEC'09), 2009, pp. 1546–1553.
[50] K. Tang, X. Yao, P.N. Suganthan, C. MacNish, Y.P. Chen, C.M. Chen, Z. Yang, Benchmark Functions for the CEC'2008 Special Session and Competition on Large Scale Global Optimization, Tech. rep., Nature Inspired Computation and Applications Laboratory, University of Science and Technology of China, China, 2007.
[51] S.-T. Hsieh, T.-Y. Sun, C.-C. Liu, S.-J. Tsai, Solving large scale global optimization using improved particle swarm optimizer, in: Proceedings of the IEEE 2008 Congress on Evolutionary Computation, Hong Kong, 2008, pp. 1777–1784.

[52] S.Z. Zhao, J.J. Liang, P.N. Suganthan, M.F. Tasgetiren, Dynamic multi-swarm particle swarm optimizer with local search for large scale global optimization, in: Proceedings of the IEEE 2008 Congress on Evolutionary Computation, Hong Kong, 2008, pp. 3845–3852.

[53] A. Zamuda, B. Bošković, Large scale global optimization using differential evolution with self-adaptation and cooperative co-evolution, in: Proceedings of the IEEE 2008 Congress on Evolutionary Computation, Hong Kong, 2008, pp. 3718–3725.