Applied Mathematical Modelling 39 (2015) 4223–4241

Contents lists available at ScienceDirect

Applied Mathematical Modelling

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

A robust parallel algorithm of the particle swarm optimization method for large dimensional engineering problems

Antonio O.S. Moraes, João F. Mitre [1], Paulo L.C. Lage *, Argimiro R. Secchi

Programa de Engenharia Química – COPPE, Universidade Federal do Rio de Janeiro, PO Box 68502, 21941-972 Rio de Janeiro, RJ, Brazil

ARTICLE INFO

Article history:
Received 27 March 2014
Received in revised form 26 November 2014
Accepted 10 December 2014
Available online 25 December 2014

Keywords:
Optimization
Particle swarm
MPI
Asynchronous parallelization
Parameter estimation
Convergence criterion

ABSTRACT

The application of the Particle Swarm Optimization (PSO) method to large engineering problems is strongly limited by the required computational cost. This limitation comes from the large number of particles needed to optimize the many-variable function, the high computational cost of its evaluation and the lack of an adequate criteria to early detect the approach of the global optimum. The first two cost sources can be mitigated by an efficient parallel implementation of the PSO method but the last one need the development of a robust convergence criterion for the algorithm. This work develops an efficient and robust optimization method by using a new convergence criterion in an asynchronous parallel implementation of PSO. In the optimization of benchmark test functions, this method showed very good performance, with parallel efficiency between 80% and 100%, and excellent robustness, always detecting the global optimum. Finally, the method was successfully applied to an actual estimation problem with 81 parameters.

1. Introduction

The numerical optimization has been subject of intense study in several fields of engineering, such as equipment and process design, process control, and parameter estimation. Two subjects of particular interest are the development of algorithms for global optimization and the optimization of large-scale engineering problems, which are computationally expensive.

The *Genetic Algorithms*, GAs [1], and the *Particle Swarm Optimization*, PSO [2], are largely used non-deterministic algorithms for global optimization, which are based on the behavior of populations (PBM – *Population Based Methods*). In the PSO, each individual particle corresponds to a potential solution of the optimization problem, located in the n-dimensional space formed by the Cartesian product of the optimization variables, and moving with its own velocity. Starting from an initial population, the algorithm consists of updating the velocity and the position of each particle in the swarm.

In its original form, the velocity update is composed by two terms: the *cognition* and the *social* terms. The *cognition* term expresses the effect of the best position recorded by each particle during its own movement and the *social* term brings about the influence of the best position among all particles in the swarm to the movement of each particle. The first important variant of the PSO was proposed by Shi and Eberhart [3] that consisted in the introduction of the *inertia weight* which ponders the influence of the particle velocity on its own motion. This new parameter balances the local and global characteristics

* Corresponding author.
E-mail addresses: paulo@peq.coppe.ufrj.br (P.L.C. Lage), arge@peq.coppe.ufrj.br (A.R. Secchi).
[1] Present address: Departamento de Engenharia Química e de Petróleo, Escola de Engenharia, Universidade Federal Fluminense, Campus Praia Vermelha, Rua Passos da Pátria, 156, Bloco D, Sala 305, Niterói, RJ 24210-240, Brazil.

http://dx.doi.org/10.1016/j.apm.2014.12.034
0307-904X/

Nomenclature

c_1	cognition parameter
c_2	social parameter
C_c	coalescence parameter
C_b	breakage parameter
E	mean value of a function or data set
f_g	the best value of the objective function known by swarm
$f_{m,i}$	the best value of the objective function known by particle i in swarm
f_d	droplet size distribution function
F	zeroth order sectional moment of the droplet volume distribution function
F_{obj}	objective function value
G	response variables
H	net particle production rate
N	permanence number counter
N_v	update counter of the inertia weight
N_p	number of particles in the swarm
N_{proc}	number of computational processors
N_{exp}	number of experiments
Q	volumetric flow rate
r_1	random number in cognition term
r_2	random number in social term
S	parallel speedup
t_{res}	hydrodynamic residence time
T	computational time
v	particle volume
$\mathbf{v}$	particle velocity vector
V_e	accident effective volume
w	inertia weight
$\mathbf{x}$	particle position vector
$\mathbf{y}$	extended particle position vector

Greek symbols

α	explanatory variables
β	model parameters
δ	errors in explanatory variables
ε	numerical tolerance
η	parallel efficiency
κ	modified PSO parameter
ξ	errors in response variables
σ	standard deviation
ς	mean number of daughters upon breakage

Subscripts

0	initial
a	absolute
$crit$	critical
f	final
g	best position in the swarm
m	best position of a particle
max	maximum
min	minimum
r	relative

Superscripts

0	initial population of the swarm
k	iteration index
n	dimension of the space

of the search. As observed by these authors, large values of the inertia weight make the swarm more exploratory, while small values favor the local search and increase the convergence speed. This variant of PSO is called in the present work as standard PSO and is given by the following equations for the updates of particle velocity and position:

$$\mathbf{v}_i^{k+1} = w\mathbf{v}_i^k + c_1 r_1 \left[\mathbf{x}_{m,i}^k - \mathbf{x}_i^k\right] + c_2 r_2 \left[\mathbf{x}_g^k - \mathbf{x}_i^k\right], \tag{1}$$

$$\mathbf{x}_i^{k+1} = \mathbf{x}_i^k + \mathbf{v}_i^k, \tag{2}$$

where $\mathbf{x}_i$ and $\mathbf{v}_i$ are, respectively, the position and the velocity of the particle i, $\mathbf{x}_{m,i}$ is the best position ever found by the particle i, $\mathbf{x}_g$ is the coordinate of the best position ever found by the swarm and the superscript k is the counter of flights (or iterations) of the algorithm, c_1 and c_2 are the cognitive and social parameters and w is the inertia weight, while r_1 and r_2 are two numbers drawn randomly in the $[0,1]$ interval for each particle at each iteration.

The PSO method has been consolidated as a good choice for global optimization problems, mainly due to its well known robustness and apparent superiority relative to other PBMs. In this context, Hassan et al. [4] compared PSO and GA methods. Their results showed that these methods had the same effectiveness, but the PSO showed better efficiency, requiring less objective function evaluations. Other advantages often cited are the easy implementation and the small amount of parameters relative to other PBM methods.

Despite of this apparent superiority, the PSO algorithm still has the large number of objective function evaluations as the main deficiency. Motivated by this limitation, many studies were focused on improving its efficiency. Clerc [5] proposed the use of a constriction factor in the velocity equation of the algorithm, in order to ensure the convergence of the particles. Biscaia et al. [6] proposed a new version of the algorithm in which the particles motion are described by the solution of an under-damped second order dynamic system, whose advantage is its unconditional stability for inertia weight values below one ($w < 1$), which is a larger region when compared with the standard PSO [7]. Chen and Zhao [8] proposed an adaptation of the swarm size based on the diversity concept, which is based on the idea that near the end of the search process, a big swarm is no longer necessary, because the diversity is small. He et al. [9] introduced a new term in the particle velocity update, named as "passive congregation", which represents the attraction on a particle exerted by others members of the swarm without being a typical social behavior. Kalivarapu et al. [10] proposed the use of digital pheromones, added as an extra term in the particle velocity update equation to emulate the action of pheromones released by insects in the search for food. Engelbrecht and van den Bergh [11] used the cooperative learning concept, an idea taken from Potter and Jong [12] who implemented this concept in their version of GA. Instead of solving a unique GA, Potter and Jong [12] partitioned the population into several groups which can cooperate through exchange of information or migration of individuals. Similarly to the previous work, Jiang et al. [13] showed an improved PSO (IPSO) algorithm, where the swarm is partitioned in sub-swarms, which can run independently the PSO algorithm or its variants. At specified *flights* (or iterations), the sub-swarms are forced to exchange information. This type of strategy was also used by Waintraub et al. [14] in their neighbor island model and by other authors in their parallel versions of the PSO, as discussed below.

The main deficiency of the PBMs is the high number of objective function evaluations required in the search process. For application in large-scale engineering problems, in which the calculation of the objective function is the most computationally expensive step, this weakness becomes critical. In order to overcome this weakness and make an efficient use of these algorithms, the parallel computation on clusters appears as a powerful and useful tool.

According to Waintraub et al. [14] and an updated search in available databases, the GAs are the PBM most exploited in parallel computations. Among such works, it is relevant to mention those of Alba and Troya and Alba et al. [15,16], that investigate the advantages that provides an asynchronous design versus a synchronous one. On the other hand, the PSO parallelization is relatively recent and still little explored in the literature. By its nature, the PSO is easily parallelizable, which makes the development of its parallel strategies an open field of studies.

The first parallelization of the PSO was developed by Schutte et al. [17], who proposed a synchronous implementation of this algorithm using the master–slave strategy, called PSPSO (Parallel Synchronous Particle Swarm Optimization). In this type of strategy, the master process performs all calculations of the algorithm except the objective function evaluation, which is performed by the slave processors. Their algorithm was tested in the optimization of a bio-mechanic system [18], showing a performance degradation when the time required for evaluating the objective function varies substantially among particles, which is caused by the synchronization requirement.

As studied by Koh et al. [19], this synchronous strategy uses efficiently the computational resources under three conditions: (i) exclusive access to machines of a homogeneous cluster, (ii) the computational time of the objective function is constant, and (iii) the parallel tasks are equally divided between processors. According to these authors, none of these conditions are, in general, satisfied, degrading the performance of the synchronous strategies. Based on these limitations, these authors proposed a parallel asynchronous strategy of the PSO, named PAPPSO (*Parallel Asynchronous PSO*). This strategy does not have a synchronization point of the particles, which is possible due the fact that the PSO algorithm does not impose the order and the number of times that the particles calculate the objective function. This feature was implemented using a FIFO (First In First Out) queue for the particles that have to be sent from master to slaves and to be received back from the slaves. The first asynchronous PSO algorithm was implemented by Venter and Sobieszczanski-Sobieski [20], which updates the particles as soon as the information are available, except for the inertia and craziness operators [21] that are only updated at the end of each flight, and the algorithm starts the next flight without waiting for all particles to be analyzed. It was clear in these

preliminary works that the asynchronous algorithms significantly outperform the synchronous algorithms in terms of parallel efficiency.

As previously described, several variants of PSO algorithms has been developed to improve its efficiency, and some of these variants were also implemented in parallel computation. For example, Kalivarapu et al. and Kalivarapu and Winer [22,23] developed synchronous and asynchronous strategies for a PSO with digital pheromones, respectively.

Also, several parallel strategies and architectures have been used, as the already mentioned master–slave strategy [24–26], the diffusion mechanism [27], the delayed exchange [28], the speculative decomposition [29], the MapReduce framework [30], the neighbor island model [14,31,32,25,26], different network topologies [32], and even hardware implementation [33]. Most of them implemented on clusters of multi-core CPUs using MPI, and, more recently, graphics processing units (GPUs) have also been used for inexpensive objective functions [34–37]. [36] exploited a high level of parallelism to fully use a 512-core GPU hardware, providing a speedup of 167 for the Rosenbrock function with 20 dimensions compared to a sequential implementation on CPU.

However, none of the above cited works solved an actual engineering optimization problem with a costly objective function in a large dimension parameter space. For this kind of problem, the convergence criterion used to stop the PSO algorithm is extremely important because one cannot afford the usage of more computational resources than what is really needed to obtain the solution within a reasonable error tolerance. Curiously, previous implementations of parallel version of the PSO did not described their convergence criterion [17,20,19] or it was simply a maximum number of objective function evaluations [14] or it was based on the permanence of the optimum of the objective function within a given small range for a few iterations [23].

In the present work, we revisited the asynchronous parallelization of the PSO, introducing the concept of a pseudo-flight and a stop criterion based on the weighted mean position of the whole population of particles. The implementation used the MPI (Message Passing Interface) library and the master–slave paradigm, being named *Asynchronous and Immediate Update Parallel PSO* (AIU-PPSO). Its parallel performance and scalability using benchmark problems were analyzed. Finally, an actual parameter estimation problem of a population balance model was successfully solved. This problem has a costly objective function and 81 parameters to be estimated.

2. The PSO algorithms

In the algorithms implemented in the present work, we used the particle velocity and position updates given by Shi and Eberhart [3], which are given by Eqs. (1) and (2). The default values of the user-defined cognition and social parameters c_1 and c_2 are 1.5. The inertia weight was proposed to be updated in a decreasing manner according to:

$$w = w_0 + (w_f - w_0)\frac{N_v}{N_{max} + N_v}, \tag{3}$$

where w_0 and w_f are the user-defined initial and final inertia weight parameters (default values are 1 and 0.1), N_{max} is the user-defined maximum number of optimum permanence and N_v is a counter used to update the inertia weight. This last parameter is zero at the beginning of the search, and it is incremented by one at the end of a flight only if the permanence criterion, defined by:

$$\left| f_g^k - f_g^{k-1} \right| < \varepsilon_a + \varepsilon_r |f_g^k|, \tag{4}$$

is satisfied. In Eq. (4), f_g is the best value of the objective function known by the swarm, k is the flight counter and ε_a and ε_r are the user-defined absolute and relative tolerances used in the permanence criterion.

The permanence of the optimum at the value f_g for $N \geqslant N_{max}$ successive iterations is commonly used as the stop criterion for the PSO algorithm. Although this is a good indication that the global optimum has been achieved, it gives no information regarding the movement of the particles in the swarm. Therefore, in order to improve the algorithm robustness, a stop criterion was introduced.

For a converged solution, all the particles in the swarm must reach to the global optimum position. However, due to the large computational cost involved, we cannot afford to wait that all particles reach it. Therefore, the basic idea behind the stop criterion is the definition of a suitable average position of the whole swarm. Due to the possibility of existence of two or more global optima, that is, optima with the same value of f but at different $\mathbf{x}$ positions, the swarm position must include the f coordinate. Therefore, consider the Cartesian $(n+1)$-dimensional space, whose elements are the extended position vectors given by

$$\mathbf{y}_i = \left[\mathbf{x}_i | \tilde{f}_i \right]^T, \tag{5}$$

where $\mathbf{x}_i$ is the position vector using optimization variables normalized in the $[0, 1]$ interval and $\tilde{f}$ is a normalized objective function given by:

$$\tilde{f}_i = \frac{f_i - \min_j(f_j^0)}{\max_j(f_j^0) - \min_j(f_j^0)}, \tag{6}$$

where $\min_j(f_j^0)$ and $\max_j(f_j^0)$ are, respectively, the minimum and maximum values of the objective function among all particles for the initial population of the swarm.

The weighted average position of the swarm about the optimal position, $\mathbf{y}_g$, is given by:

$$\bar{\mathbf{y}} = \sum_{i=1;i\neq g}^{N_p} \bar{\omega}_i \mathbf{y}_i, \tag{7}$$

where N_p is the number of particles in the swarm and $\bar{\omega}_i$ is the weight of particle i. The need for a different weight to each particle position comes from the way the PSO is employed. Initially, the particles are randomly distributed in the search domain and they are moved with a random velocity component. If a local or global optimum is present, it quickly gathers the nearby particles but it slowly attracts the particles located far away from it. Therefore, the weight of the particle position should be inversely proportional to its distance to the global optimum. After some initial tests reported elsewhere [38], a good definition of this weight is:

$$\bar{\omega}_i = \frac{\frac{1}{\|\mathbf{y}_i - \mathbf{y}_g\|}}{\sum_{i=1;i\neq g}^{N_p} \frac{1}{\|\mathbf{y}_i - \mathbf{y}_g\|}}. \tag{8}$$

This definition heavily weights the particles near the optimal position, taking indirectly into account the swarm spread in this $(n+1)$-dimensional space.

Thus, the stop criterion is based on the displacement of the weighted average position of the swarm, $\bar{\mathbf{y}}$, between two flights for which the permanence criterion was successfully satisfied N_{min} times (the user-defined minimum number of iterations with the permanence of the optimum at the same position), and is given by:

$$\left\|\bar{\mathbf{y}} - \bar{\mathbf{y}}^{old}\right\| < \varepsilon_a, \tag{9}$$

where $\bar{\mathbf{y}}$ and $\bar{\mathbf{y}}^{old}$ are the weighted average positions of the swarm at the current iteration where the permanence criterion has been successively satisfied N_{min} times and the last iteration where this has also happened. Therefore, close to convergence, when the permanence criterion is satisfied for all iterations, the stop criterion is evaluated at every N_{min} iterations.

All the norms used in the above equations are the normalized Euclidean norm, defined by:

$$\|\mathbf{y}\| = \left[\frac{1}{n}\sum_{i=1}^{n} y_i^2\right]^{\frac{1}{2}}. \tag{10}$$

2.1. Serial PSO algorithm

The implementation of serial PSO algorithms is quite simple. The coordinates of the parameter space are made dimensionless upon linear transformation to the $[0,1]^n$ n-dimensional cube. After initialization of the positions and velocities of the particles, usually randomly, there is a flight loop where all particles have their velocities and positions updated according to the best values of the objective function calculated for each particle and for the swarm. At the end of each flight, the permanence criterion given by Eq. (4) is verified. When this test is consecutively satisfied N_{min} times, the stop criterion given by Eq. (9) is then tested. Algorithm 1 shows the pseudo-code of the serial PSO algorithm for a minimization problem. In this pseudo-code, Max_{aval} is the user-defined maximum number of objective function evaluations, the *rand*() function give a random number in the range $[0,1]$ with uniform distribution and N_p is the number of particles.

2.2. Parallel PSO algorithms

The parallel algorithms were developed using a master–slave strategy for data communication among the available processors, N_{proc}. In this strategy, the master processor coordinates all steps of the algorithm and manages all the communications with the $N_{slaves}(= N_{proc} - 1)$ slaves processors, which calculate the objective function.

2.2.1. Parallel synchronous algorithm (IU-PPSO)

The parallel synchronous algorithm of PSO developed in the present work, named *Immediate Update Parallel PSO* (IU-PPSO), can be seen as a simple extension of the serial algorithm. It is essentially identical to the serial algorithm, with the difference that the evaluations of the objective function are performed in parallel. The *point-to-point MPI_Send()* and *MPI_Recv()* communication functions of the MPI library [39] were used for data communication between processors. Algorithm 2 shows the pseudo-code of the algorithm performed by the master processor for a minimization problem. The slaves only perform the objective function calculation, as showed by Algorithm 3.

As shown in Algorithm 2, the initialization is carried out by evaluating the objective functions for all particles using the available slaves, N_{slaves}, and the calculation of some control variables. Afterwards, the algorithm start the optimization loop. At the end of each flight, there is a synchronization of all slaves before testing the permanence of the optimum.

2.2.2. Parallel asynchronous algorithm (AIU-PPSO)

The synchronous algorithm shows a weakness on its parallel strategy: the need of synchronization makes the algorithm limited by the slower slave. In other words, at the synchronization point the master must wait that all slaves finish their work to continue the algorithm letting any available slave in an idle waiting state.

In order to overcome this limitation, an asynchronous strategy of parallelization was implemented, where there is no synchronization point. Thus, the concept of a flight is destroyed, which brings about the definition of a pseudo-flight. This consists of defining a number of objective function evaluations after which the master processor exits the optimization step to test the convergence criterion. In synchronous algorithms, the number of evaluations of a flight is equal to the number of particles. In the asynchronous algorithm of Venter and Sobieszczanski-Sobieski [20], this number is not constant because the authors assume that a flight is finished only when all particles have concluded their function evaluations of a given iteration, even if some particles had finished posterior iterations. Koh et al. [19] did not use the concept of a flight in their asynchronous algorithm. In the present work, the number of particles was assumed as the number of evaluations of a pseudo-flight, independently of which particles made these evaluations, and the master tests the optimum permanence and stop criteria without stopping the work that is being done by the slaves. Therefore, the only time loss in the parallelization occurs when a slave wants to communicate with the master when it is executing the optimization step. In this situation, the slave becomes momentarily idle. However, when the objective function is the most costly step, this loss is negligible.

In order to obtain asynchrony, the particles are arranged in a FIFO (*First In First Out*) queue. Since the computational time of the objective function varies, the order of the particles in the queue also varies and, therefore, some particles may not contribute to the optimization in a pseudo-flight. Besides, the slaves may perform different number of function evaluations, which make the asynchronous algorithm unable to maintain the consistency described by Schutte et al. [17].

The AIU-PPSO implementation is similar to the IU-PPSO pseudo-code shown in Algorithms 2 and 3. The only difference is the optimization step (pseudo-flight loop) performed by the master processor in the AIU-PPSO which is shown in Algorithm 4.

3. Algorithm evaluation

3.1. Software and hardware environments

The implementation of the PSO algorithms were developed using the GNU C compiler, versions 4.1.2 and 4.4.3, and the MPI (Message Passing Interface) library, OpenMPI versions 1.2.6 and 1.4.2. The hardware platform was a MIMD parallel computer with distributed memory and 20 Gbits/s DDR Infiniband network. The cluster nodes were dual Quad-Core AMD 2356 processors with 16 GB of memory.

3.2. Benchmark test functions

Several benchmark test functions were used to evaluate the algorithms. In this work, the results are presented for the two-dimensional Alpina and Levy functions and for the multidimensional Ackley, Schwefel [40] and H1 [19] functions, defined by Eqs. (11)–(15), respectively. All of these functions have many local extrema, but only one global maximum or minimum, which makes them good choices for testing global optimization algorithms.

Alpina function:

$$F_{obj}(\mathbf{x}) = \sqrt{x_1 x_2}\,\sin(x_1)\sin(x_2), \quad \mathbf{x} \in [0,10]^2, \tag{11}$$

whose global maximum is 7.8856 occurring at $\mathbf{x} = (7.917, 7.917)$.

Levy function:

$$F_{obj}(\mathbf{x}) = \left\{\sum_{i=1}^{5} i\cos\left[(i-1)x_1 + 1\right]\right\}\left\{\sum_{i=1}^{5} i\cos\left[(i+1)x_2 + 1\right]\right\} + (x_1 + 1.42513)^2 + (x_2 + 0.80032)^2, \quad \mathbf{x} \in [-10,10]^2, \tag{12}$$

whose global minimum is -176.1376 at $\mathbf{x} = (-1.3068, -1.4248)$.

Ackley function:

$$F_{obj}(\mathbf{x}) = -20\exp\left[-0.2\sqrt{\sum_{i=1}^{n}\frac{x_i^2}{n}}\right] - \exp\left[\sum_{i=1}^{n}\frac{\cos 2\pi x_i}{n}\right] + 20 + e^1 \tag{13}$$

for $\mathbf{x} \in [-32.68, 32.68]^n$, whose global minimum is at the origin.

Schwefel function:

$$F_{obj}(\mathbf{x}) = 418n - \sum_{i=1}^{n} x_i \sin\left[\sqrt{(|x_i|)}\right], \quad \mathbf{x} \in [-500, 500]^n, \tag{14}$$

whose global minimum is at $\mathbf{x} = [420.9687, 420.9687]^n$.

H1 function:

$$F_{obj}(\mathbf{x}) = \frac{\sin^2\left[x_1 - \frac{x_2}{8}\right] + \sin^2\left[x_2 + \frac{x_1}{8}\right]}{1 + \sqrt{(x_1 - 8.6998)^2 + (x_2 - 6.7665)^2}}, \quad \mathbf{x} \in [-100, 100]^n, \tag{15}$$

whose global minimum is at $\mathbf{x} = [8.6998, 6.7665]$.

3.3. Evaluation metrics

The parallel algorithms were evaluated using the speedup and parallel efficiency concepts. The fair speedup, S, is defined as the ratio between the computational time spent by the serial algorithm and that one spent by the parallel algorithm:

$$S = \frac{T_s}{T_p}, \tag{16}$$

where T_s and T_p are the serial and the parallel computational times.

The fair speedup shows the effective gain that the parallel algorithm provides over the serial one. In the ideal case, it is equal to N_{proc}, that is the number of computational processors used in the parallel computation. The parallel efficiency is defined as the ratio between the effective speedup (fair speedup) and the ideal speedup, as defined by:

$$\eta = \frac{S}{N_{proc}}. \tag{17}$$

For each benchmark test function, each algorithm was executed N_{run} times, using different seeds to randomly initialize the populations. The computational time used in Eq. (16) is the mean computational time of the N_{run} runs. As each numerical experiment is independent, a relation to calculate the standard deviation of the speedup is necessary. This relation is provided by Eq. (18), whose demonstration is provided by Moraes [38].

$$\sigma_S^2 = \frac{\left[\sigma_{T_s}^2 + E(T_s)^2\right]\left[\sigma_{T_p}^2 + E(T_p)^2\right]}{E(T_p)^4} - \left[\frac{E(T_s)}{E(T_p)}\right]^2 + \frac{\sigma_{T_p}^2}{E(T_p)^4}\left[2\sigma_{T_s}^2 - \sigma_{T_p}^2\frac{E(T_s)^2}{E(T_p)^2}\right], \tag{18}$$

where $E(\psi)$ and σ_ψ are, respectively, the mean and the standard deviation of the variable ψ.

4. Model used for parameter estimation

The parameter estimation problem in a large dimensional space considers a *Population Balance Model* for the water-in-oil emulsion flow in a duct with a valve-type accident [41]. This model describes the evolution of the droplet size distribution function (PSDF), f_d, including droplet breakage and coalescence, whose models have parameters that need to be estimated.

The evolution of f_d is given by the solution of the population balance equation (PBE). As only the normalized PSDFs at the outlet and inlet of the test section were known, Mitre et al. [41] decided to use a simplified steady-state Lagrangian model in the form:

$$\frac{Df_d}{Dt} = H(\{f_d\}; \nu), \tag{19}$$

where H is the droplet net production rate by breakage and coalescence, which are defined elsewhere [42,41].

This simplified model assumes that the phases are incompressible and all droplets move with the continuous phase velocity, which allows that the final distribution can be obtained by integrating Eq. (19) in t up to the mean hydrodynamic residence time, t_{res}, which is defined by

Table 1
User defined parameters used for the benchmark problems.

Test function	w_0	w_f	N_p	N_{min}	N_{max}	$\mathtt{Max}_{aval}$
Alpina	1	0.1	31	20	80	10^5
Ackley (2D)	1	0.1	31	20	80	10^5
Levy	1	0.1	31	20	80	10^5
Schwefel (2D)	1	0.1	150	20	80	10^7
Ackley (32D)	1	0.1	1000	40	160	10^7
H1	0.75	0.01	50	10	80	10^7

$c_1 = c_2 = 1.5$, $\varepsilon_a = \varepsilon_r = 10^{-4}$.

Table 2
Computational costs of the objective functions.

Test function	CPU time [s]	
	Mean	Standard deviation
Alpina	1.1581	0.3549
Levy	1.2079	0.3616
Ackley (2D)	1.2493	0.3599
Schwefel (2D)	0.02537	0.006893
Ackley (32D)	0.03179	0.005794

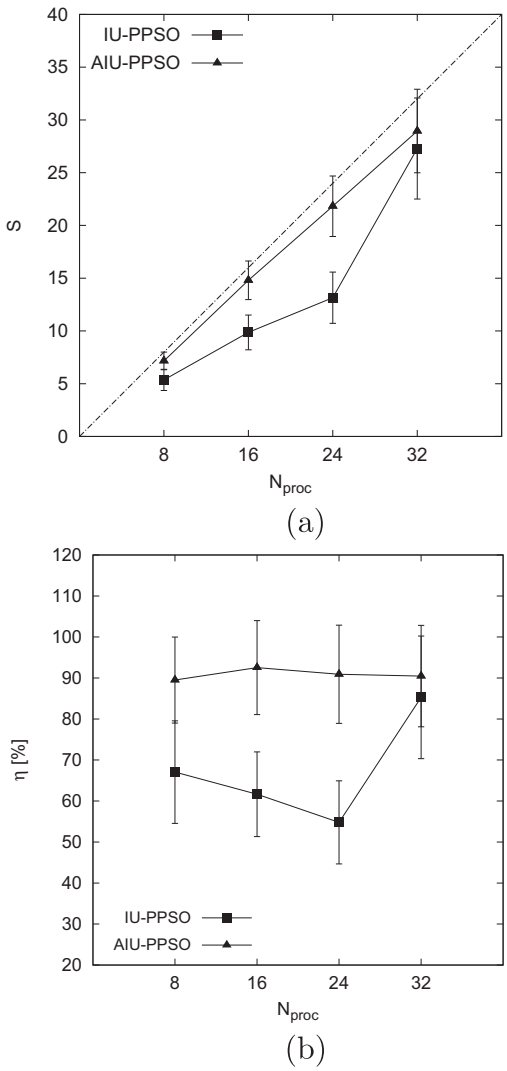

Fig. 1. Alpina test function: (a) speedup and (b) efficiency of the parallel algorithms.

$$t_{res} = \frac{V_e}{Q}, \tag{20}$$

where V_e is the accident effective volume and Q is the volumetric flow of the emulsion. Although Q is known for each experiment, the accident effective volume can be considered another parameter of the model, whose value is different for each experimental run. Lage et al. [43] showed that the droplets break mainly nearby the accident, in a region whose volume is somewhat constant and may be approximated by the geometry of the accident [41]. Therefore, V_e was firstly considered

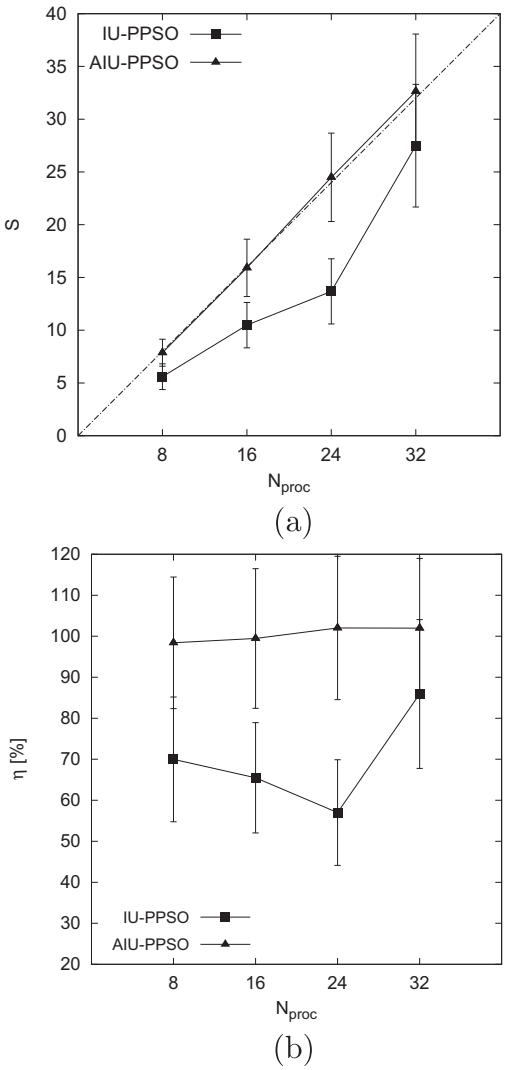

Fig. 2. Levy test function: (a) speedup and (b) efficiency of the parallel algorithms.

to be equal to this geometrical approximation. Afterwards, this assumption was verified by estimating a different value of V_e for each run.

The coalescence and breakage models are described by Mitre et al. [41], but they introduce three parameters to be estimated: C_b, C_c, and ς. Available experimental data consist of inlet and outlet PSDFs of 78 experiments. Therefore, the total number of parameters is 81, that consist of the three model parameters cited above and 78 values of V_e, one for each experiment. More details of the model or its numerical solution can be found elsewhere [44,41].

5. Results

5.1. Benchmark problems

In order to evaluate the performance and scalability of the parallel algorithms, they were applied in the optimization of the benchmark test functions defined in Section 3.2. Table 1 shows the user defined parameters used for these problems. In order to emulate a larger problem with a varying computational cost, a random delay was added to the evaluation of each of these objective functions. Table 2 shows the resulting computational times for Alpina, Levy, Ackley and Schwefel functions with two optimization variables and for the Ackley function with 32 variables. For all the results presented in the following, all runs of the PSO algorithms were successful in determining the optimum within the required tolerance, showing that the convergence criterion proposed in Section 2 is quite robust.

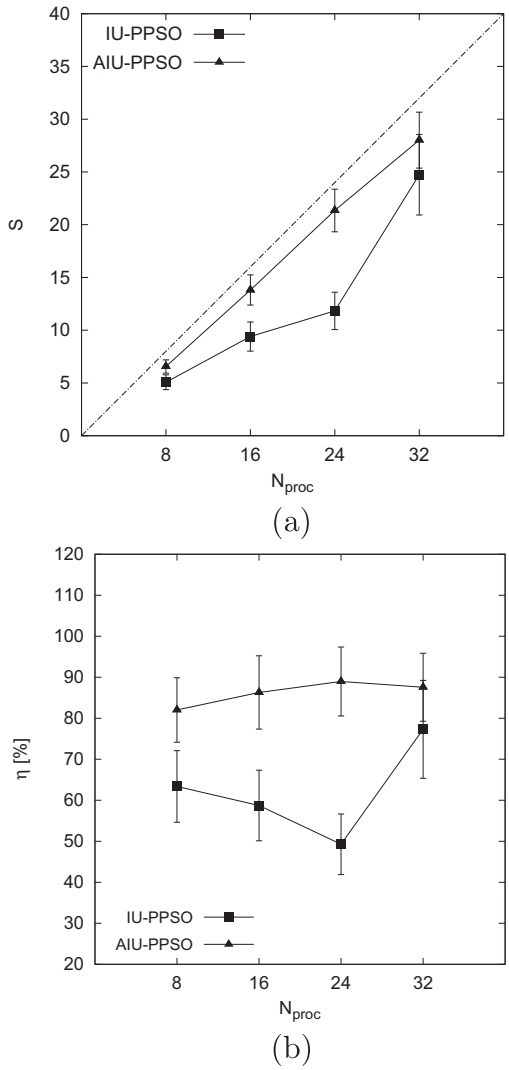

Fig. 3. Ackley two-dimensional test function: (a) speedup and (b) efficiency of the parallel algorithms.

Figs. 1–3 show the speedup (S) and efficiency (η) for the synchronous (IU-PPSO) and asynchronous (AIU-PPSO) algorithms in the optimization of Alpina, Levy and Ackley 2D functions, respectively. The vertical bars in the graphs correspond to the standard deviations about the mean, calculated for 50 independent numerical experiments using different initial guesses. The diagonal line in the speedup plots refers to the ideal speedup.

The asynchronous algorithm showed nearly linear speedup with mean parallel efficiency around or above 90% for these three test functions, while the efficiency of the synchronous algorithm decreased with N_{proc} for $N_{proc} \leqslant 24$. For 32 computational processors (31 slave processors), larger parallel efficiencies of the synchronous algorithm were obtained for the three benchmark functions. In these cases, the number of slave processors was equal to the number of the particles in the swarm. In this situation, as long as the computational time of the objective function is large and relatively insensitive to the parameter values, the idleness of the slaves is minimal, because all particles of the swarm can be computed by the slaves at the same time in each flight or algorithm iteration. Therefore, the loss of efficiency due to the idleness of the slaves in this situation is caused by the range of computational time for the objective function evaluation.

Figs. 4 and 5 show the speedup (S) and efficiency (η) for the synchronous (IU-PPSO) and asynchronous (AIU-PPSO) algorithms in the optimization of Schwefel 2D and Ackley 32D functions, respectively. Again, the vertical bars in the graphs correspond to the standard deviation about the mean for 100 independent numerical experiments. As shown in Table 1, 150 particles were used in the optimization process for the Schwefel function and 1000 particles were employed for the Ackley function optimization.

The asynchronous algorithm showed linear speedup with mean parallel efficiency close to 100% for these two test functions. On the other hand, the performance of the synchronous algorithm degraded when the number of processors increases.

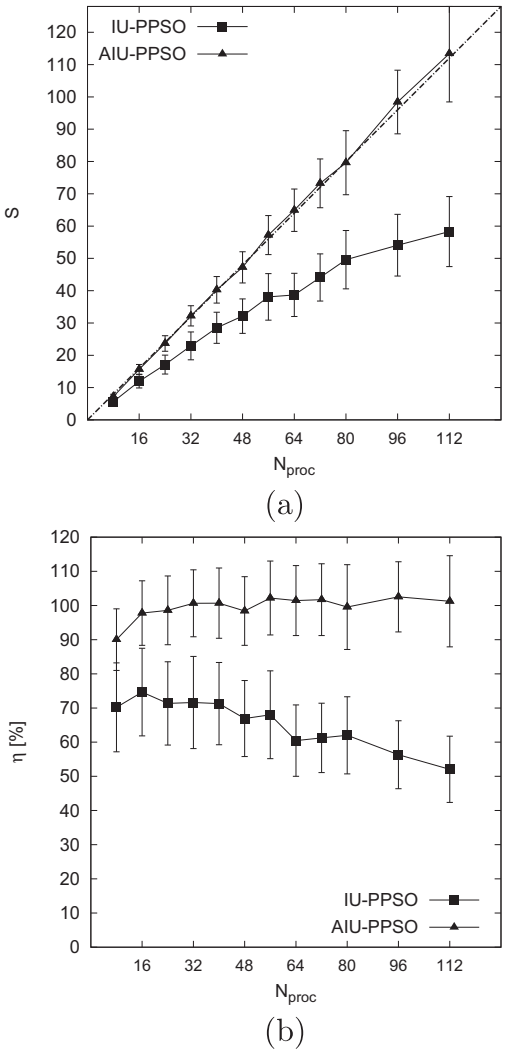

Fig. 4. Schwefel two-dimensional test function: (a) speedup and (b) efficiency of the parallel algorithms.

The higher efficiency of the AIU-PPSO for all tested cases is due to the nature of the parallelization. As commented in Section 2.2.2, the asynchrony of the algorithm reduces the loss of computational time of the slaves waiting to communicate with the master when it is executing the optimization step. When the objective function computational time is large, this idleness is quite small. The synchronization point in the IU-PPSO makes this loss intrinsic to the algorithm, being affected by the mismatch between the numbers of slave processors and particles and by the range of computational time for the evaluation of the objective function.

Mean speedup values larger than the corresponding ideal values are shown in Figs. 2, 4, 5. Due to the large standard deviations of these numerical experiments, we cannot state that the AIU-PPSO algorithm presented a superlinear speedup, even though this might be possible due to the synergetic effect of the asynchronous evaluation of the particles and the pseudo-flight concept or even a more efficient utilization of computational resources, such as increased availability of RAM and cache memory for objective function calculation by the slaves processors.

Table 3 shows the computational time for the H1 function with an added random time delay to increase the range of computational time for its evaluation, as done by Koh et al. [19]. In other words, the CPU time is randomly distributed within the ranges listed in Table 3. Fig. 6 shows the speedup and the efficiency of the IU-PPSO and AIU-PPSO in the optimization of these problems. Again, the AIU-PPSO algorithm achieved almost linear speedup with parallel efficiency around 90%, while the IU-PPSO performance degraded with the increase of the number of processors. Unlike the results presented by Koh et al. [19], the CPU time variability of the objective function did not affect the performance of the parallel algorithms developed in the present work.

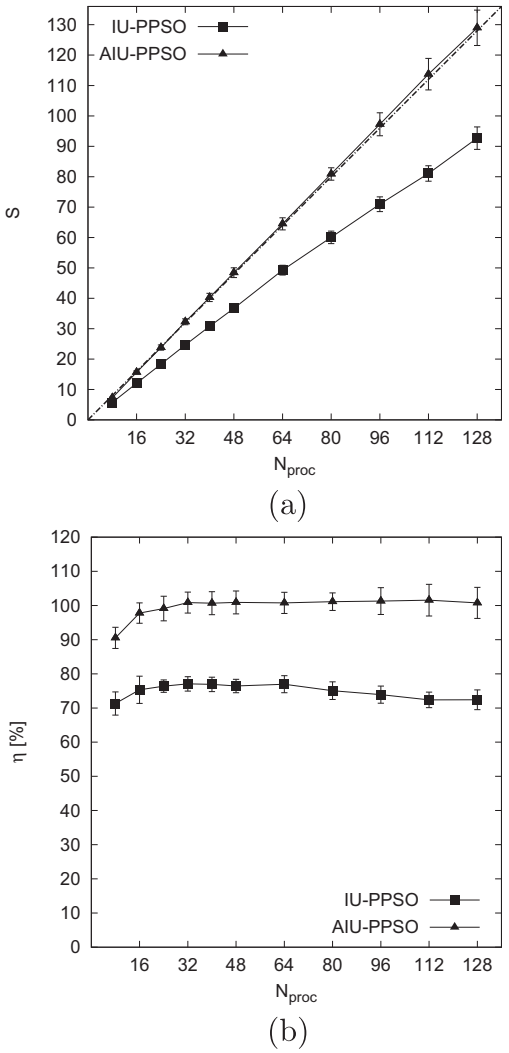

Fig. 5. Ackley 32-dimensional test function: (a) speedup and (b) efficiency of the parallel algorithms.

Table 3
Computational costs of the H1 function.

Time delay	CPU time [s]	
	Mean	Range
None	0.5	[0.5, 0.5]
Up to 20%	0.55	[0.5, 0.6]
Up to 50%	0.625	[0.5, 0.75]

5.2. Parameter estimation problem

In order to simplify the notation, let us assume that the model provides an explicit relation given by:

$$\tilde{G}_i \cong G(\alpha_i; \beta), \tag{21}$$

where i is the experiment index, $\tilde{G}_i$ and α_i are, respectively, the values of the response and explanatory variables, both experimentally obtained, $G(\alpha_i; \beta)$ are the values of the response variables according to the model and β are the model parameters.

Assuming a perfect model, the error, ξ_i, of the model prediction for the ith experiment is:

$$\xi_i = G(\alpha_i + \delta_i; \beta) - \tilde{G}_i, \tag{22}$$

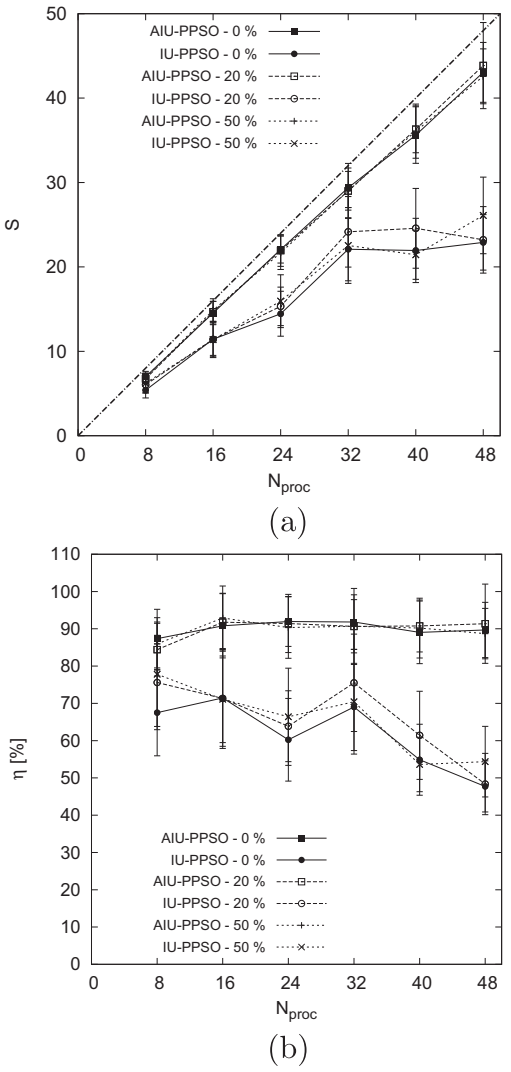

Fig. 6. H1 test function with different ranges of computational time: (a) speedup and (b) efficiency of the parallel algorithms.

where δ_i are the unknown errors of the explanatory variables, given by:

$$\delta_i = \bar{\alpha}_i - \alpha_i, \tag{23}$$

where $\bar{\alpha}_i$ is the actual value of the explanatory variable or, in other words, the value of this variable when the objective function is minimal.

The objective function for the parameter estimation problem by the ODRPACK95 optimizer [45], which has been employed by Mitre et al. [41], is given by:

$$F_{obj} = \sum_{i=1}^{N_{exp}} \left[\xi_i^T W_{\xi_i} \xi_i + \delta_i^T W_{\delta_i} \delta_i\right], \tag{24}$$

where N_{exp} is the number of experiments, W_{ξ_i} is the weight of each response variable and W_{δ_i} is the weight of each explanatory variable. The AIU-PPSO algorithm used the objective function given by Eq. (24) but with no errors in the explanatory variables, that is, $\delta_i = 0, \ \forall i$.

The CPU time of the objective function given by Eq. (24) was estimated on the cores of the cluster nodes. For this task, 64 simulations were carried out using parameter values uniformly distributed over the domain of the three main parameters of the model (C_c, C_b and ς). The mean CPU time was 1138.93 s with a standard deviation of 187.65 s. For this problem, 78 experiments were considered, which add the V_e value as a parameter for each experimental run, enlarging the dimension

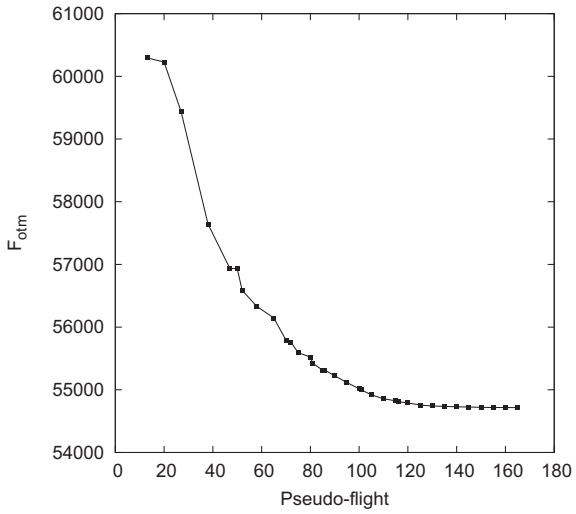

Fig. 7. Evolution of the objective function value along the pseudo-flights of the AIU-PPSO run.

Table 4
Comparison between indicator function I for the optimization cases.

Case k	I_k	I_1/I_k
1	0.305	1.000
2	0.316	0.965
3	0.254	1.201
4	0.267	1.142

of the parameter space to 81. The AIU-PPSO algorithm was used with the default values of the user defined parameters ($w_0 = 1,\ w_f = 0.1$ and $c_1 = c_2 = 1.5$), 2000 particles, and $\varepsilon_r = \varepsilon_a = 10^{-3}$.

Fig. 7 shows the evolution of the objective function value along the pseudo-flights, each one consisting of 2000 objective function evaluations. The swarm converged after 165 pseudo-flights and 330,119 objective function evaluations, taking 66 days to be completed. The application of serial algorithms of PSO or other PBMs would be unfeasible, since the computational cost of evaluating the objective function 330,119 times, would take about 12 years in the same processor cores.

5.2.1. Comparison between AIU-PPSO and ODRPACK95 results

The optimal values obtained with AIU-PPSO were compared with those obtained with the ODRPACK95 optimizer [45] in Mitre et al. [41]. Besides the parameter values of the breakage and coalescence models, the main results of the parameter estimation problem is the fitness of the size distribution predicted by the population balance model at the outlet. Since each optimizer used a different objective function, an indicator function, defined by Eq. (25), was used to allow this comparison when the errors in the explanatory variables are considered or not in the optimization.

$$I = \sum_{i=1}^{N_{exp}} \sum_{j=1}^{N_c} \left[\frac{F_{i,j} - \tilde{F}_{i,j}}{\alpha_d^2} \right], \tag{25}$$

where i is the experiment index, j is the index of the droplet class, N_c is the number of classes, $\tilde{F}_{i,j}$ are the experimental values of the zeroth-order sectional moment of the volume distribution function at the outlet, $F_{i,j}$ are the corresponding values predicted by the model and α_d is the first moment of the volume distribution function which is equal to the phase fraction of the dispersed phase.

The indicator function was evaluated for the following cases:

1. ODRPACK95 results with no errors in the explanatory variables as given by Araujo [44];
2. ODRPACK95 results considering the errors in all explanatory variables as given by Araujo [44];
3. AIU-PPSO results with no errors in the explanatory variables;
4. ODRPACK95 results considering the errors in all explanatory variables but seeded with the optimum values obtained in item 3 above for the 81 parameters.

Table 5
Estimated parameters of the model for each evaluated case.

Case	C_c	C_b	ς
1	$(1.00 \pm 0.05) \times 10^{-2}$	$(0.98 \pm 0.06) \times 10^{-2}$	26.0 ± 0.9
2	$(1.05 \pm 0.23) \times 10^{-2}$	$(1.06 \pm 0.80) \times 10^{-2}$	26.9 ± 9.4
3	1.90×10^{-2}	1.07×10^{-2}	31.1
4	$(1.88 \pm 0.06) \times 10^{-2}$	$(1.08 \pm 0,7) \times 10^{-2}$	32.7 ± 16.8

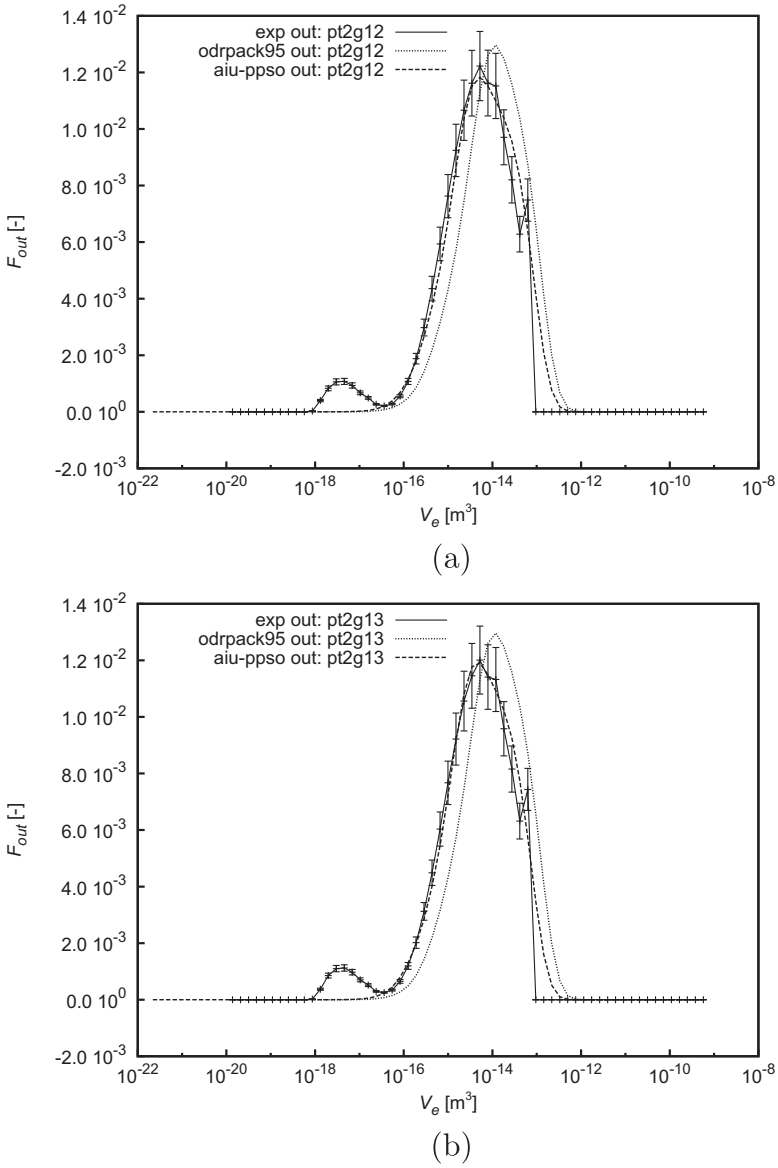

Fig. 8. Experimental and simulated PSD at the outlet using the optimum parameters found by different optimizations: experiments (a) pt2g12 and (b) pt2g13.

Table 4 shows the values of the indicator function for these four cases. As cases 1 and 3 can be directly compared because they used the same objective function, it can be said that the quality of the results obtained by AIU-PPSO is around 20% better than that obtained with ODRPACK95. The results for case 4 also shows that the solution refinement obtained with the local

optimizer (ODRPACK95) using initial values from the global optimizer (AIU-PPSO) had a small impact in the indicator function. Table 5 shows the estimated values for the parameters of the breakage and coalescence models for these optimizations. For the ODRPACK95 results, the 95% confidence intervals of the parameter values are also shown.

Finally, Fig. 8 shows the PSDFs obtained experimentally and using the model with the parameter values estimated by ODRPACK95 (case 0) and AIU-PPSO (case 3) for two experimental runs (*pt2g12* and *pt2g13* from Araujo [44], illustrating the best quality of the AIU-PPSO results.

6. Conclusions

For all cases analyzed, the AIU-PPSO algorithm provided the best parallel performance and scalability, with linear speedup and mean parallel efficiency around or above 90%, even when using all the 128 processors of the available cluster. On the other hand, the performance of the synchronous algorithm IU-PPSO degraded with the increase in the number of processors. This loss is associated with the synchronization point of this algorithm, which makes its speed to be limited by the slower slave in each flight (or iteration) of the algorithm. The AIU-PPSO algorithm did not show any sensitivity to the variability of the computational cost of the objective function.

Using the proposed convergence criterion, the PSO algorithms became quite robust, being able to successfully determine the global optimum within the required tolerance for all runs of all test cases.

For the estimation parameter problem, the AIU-PPSO made possible the optimization of an engineering problem with 81 parameters. Due to the computational time of the objective function, it was impractical to use serial PSO or other serial PBM to solve it. Besides, the results obtained by AIU-PPSO were 20% better than those obtained by ODRPACK95, that shows the advantage in use the global optimizers over local ones to parameter estimation problems.

Algorithm 1. Pseudo-code of the PSO serial implementation

Require: $w_0,\ w_f,\ c_1,\ c_2,\ N_p,\ N_{min},\ N_{max},\ \mathtt{Max}_{aval}$
  for $(i = 1 \rightarrow N_p)$ **do**
    $\mathbf{x}_{m,i} \leftarrow rand()$
    $\mathbf{x}_i \leftarrow \mathbf{x}_{m,i}$
    $f_{m,i} \leftarrow F_{obj}(\mathbf{x}_i)$
  end for
  $(f_g, j) \leftarrow (\min_i f_{m,i}, i)$
  $\mathbf{x}_g \leftarrow \mathbf{x}_j$
  $k \leftarrow 0$
  $N \leftarrow 0$
  $N_v \leftarrow 0$
  Calculate $\bar{\mathbf{y}}$ using Eq. (7)
  $optimum \leftarrow f_g$
  $N_{aval} \leftarrow N_p$
  while $(N < N_{max}$ and $N_{aval} < \mathtt{Max}_{aval})$ **do**
    Update w with Eq. (3)
    for $(i = 1 \rightarrow N_p)$ **do**
      Update $\mathbf{x}_i$ and $\mathbf{v}_i$, Eqs. (2) and (1)
      $F_i = F_{obj}(\mathbf{x}_i)$
      $N_{aval} \leftarrow N_{aval} + 1$
      if $(F_i < f_{m,i})$ **then**
        $f_{m,i} \leftarrow F_i$
        $\mathbf{x}_{m,i} \leftarrow \mathbf{x}_i$
      end if
      if $(F_i < f_g)$ **then**
        $f_g \leftarrow F_i$
        $\mathbf{x}_g \leftarrow \mathbf{x}_i$
    end if
  end for
  Performs ***Permanence and stop criteria***
  $k \leftarrow k + 1$
end while

Permanence and stop criteria, Eqs. (4) and (9):

if $\left(|f_g - optimum| < \varepsilon_a + \varepsilon_r |f_g|\right)$
then
  $N \leftarrow N + 1$
  $N_v \leftarrow N_v + 1$
  if $\left(\frac{N}{N_{min}} \in \mathbb{N}\right)$ **then**
    $\bar{\mathbf{y}}^{old} \leftarrow \bar{\mathbf{y}}$
    Calculate $\bar{\mathbf{y}}$ using Eq. (7)
    if $(\|\bar{\mathbf{y}} - \bar{\mathbf{y}}^{old}\| < \varepsilon_a)$ **then**
      Stop while loop and write results
    end if
  end if
else
  $N \leftarrow 0$
  if $(f_g < optimum)$ **then**
    $optimum = f_g$
  end if
end if

Algorithm 2. Master processor pseudo-code of the IU-PPSO

Require: $w_0,\ w_f,\ c_1,\ c_2,\ N_p,\ N_{proc}$,
$N_{min},\ N_{max}$ and $\mathtt{Max}_{aval}$
$N_{slaves} = N_{proc} - 1$
for $(i = 1 \rightarrow N_p)$ **do**
$\mathbf{x}_{m,i} \leftarrow rand()$
$\mathbf{x}_i \leftarrow \mathbf{x}_{m,i}$
if $(i < N_{slaves})$ **then**
Send $\mathbf{x}_i$ to slave i
else
Receive F_{obj} concerning the particle j from slave p
$F_{m,j} \leftarrow F_{obj}$
Send $\mathbf{x}_i$ to slave p
end if
end for
synchronizes the slaves
$(f_g, j) \leftarrow (\min_i f_{m,i}, i)$
$\mathbf{x}_g \leftarrow \mathbf{x}_j$
$k \leftarrow 0$
$N \leftarrow 0$
$N_v \leftarrow 0$
Calculate $\bar{\mathbf{y}}$ using Eq. (7) Calculate $\bar{\mathbf{y}}$ using Eq. (7)
$optimum \leftarrow f_g$
$N_{aval} \leftarrow N_p$
Performs the **Optimization step**

Optimization step (flight loop)

while $(N < N_{max}$ and $N_{aval} < \mathtt{Max}_{aval})$ **do**
Update w with Eq. (3)
for $(i = 1 \rightarrow N_p)$ **do**
if $(i < N_{slave})$ **then**
Update $\mathbf{x}_i$ and $\mathbf{v}_i$. Eqs. (2) and (1)
Send $\mathbf{x}_i$ to slave i
else
Receive F_{obj} concerning the particle j from slave p
$N_{aval} \leftarrow N_{aval} + 1$
if $(F_{obj} < f_{m,j})$ **then**
$f_{m,j} \leftarrow F_{obj}$
$\mathbf{x}_{m,i} \leftarrow \mathbf{x}_j$
end if
if $(F_{obj} < f_g)$ **then**
$f_g \leftarrow F_{obj}$
$\mathbf{x}_g \leftarrow \mathbf{x}_j$
end if
Update $\mathbf{x}_i$ and $\mathbf{v}_i$, Eqs. (2) and (1)
Send $\mathbf{x}_i$ to slave p
end if
end for
Slaves synchronization
Performs **Permanence and stop criteria** (see Algorithm 1)
end while

Algorithm 3. Slave processor pseudo-code of the IU-PPSO

while (not received the stop order from master) **do**
Receive $\mathbf{x}_i$ from master
$F_{obj} = F(\mathbf{x}_i)$
Send F_{obj} to master
end while

Algorithm 4. Master processor pseudo-code of the optimization step in the AIU-PPSO

Optimization step (pseudo-flight loop)
while $(N < N_{max}$ and $N_{aval} < \mathtt{Max}_{aval})$ **do**
Update w with Eq. (3)
Receive F_{obj} concerning the particle j from slave p
$N_{aval} \leftarrow N_{aval} + 1$
Insert the particle j in the FIFO queue
if $(F_{obj} < f_{m,j})$ **then**
$f_{m,j} \leftarrow F_{obj}$
$\mathbf{x}_{m,j} \leftarrow \mathbf{x}_j$
end if
if $(F_{obj} < f_g)$ **then**

(*continued on next page*)

$f_g \leftarrow F_{obj}$
$\mathbf{x}_g \leftarrow \mathbf{x}_j$
end if
Remove from the queue the next particle (i) to be calculated
Update $\mathbf{x}_i$ and $\mathbf{v}_i$ using Eqs. (2) and (1)
Send $\mathbf{x}_i$ to slave p
if (End of pseudo-flight) **then**
Performs **Permanence and stop criteria** (see Algorithm 1)
end if
end while

Acknowledgments

Antonio de O. S. Moraes acknowledges the financial support from CNPq, Grant No. 140951/2012-1. Argimiro R. Secchi acknowledges the financial support from CNPq, Grants 304907/2009-0 and 480040/2010-9. Paulo L. C. Lage acknowledges the financial support from CNPq, Grant Nos. 302963/2011-1 and 476268/2009-5 and from FAPERJ, Grant No. E-26/111.361/2010.

References

[1] J.H. Holland, Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence, first ed., MIT Press Cambridge, MA, USA, 1992.
[2] R. Eberhart, J. Kennedy, Particle swarm optimization, Proc. IEEE Int. Conf. Neural Netw. (1995) 1942–1948
[3] Y. Shi, R.C. Eberhart, Parameter selection in particle swarm optimization, in: Evolutionary Programming VII: Proceedings of the EP98, Springer-Verlag, New York, 1998.
[4] R. Hassan, B. Cohanim, O. Weck, G. Venter, A comparison of particle swarm optimization and the genetic algorithm, Technical Report, Massachusetts Institute of Technology, Cambridge, MA, 02139, 2004.
[5] M. Clerc, The swarm and the queen: toward a deterministic and adaptive particle swarm optimization, Evol. Comput. CEQ 99 (1999) 1951–1957.
[6] J.E.C. Biscaia, M. Schwaab, J.C. Pinto, Um novo enfoque do método do enxame de partículas, Proceedings of the Workshop em Nanotecnologia e Computação Inspirada na Biologia, 2004.
[7] F. van den Bergh, A. Engelbrecht, A study of particle swarm optimization particle trajectories, Inf. Sci. 176 (2006) 937–971.
[8] D. Chen, C. Zhao, Particle swarm optimization with adaptive population size and its application, Appl. Soft Comput. 9 (2009) 39–48.
[9] S. He, Q. Wu, J. Wen, J. Saunders, R. Paton, A particle swarm optimizer with passive congregation, Biosystems 78 (2004) 135–147.
[10] V.K. Kalivarapu, J.-L. Foo, E.H. Winer, Improving solution characteristics of particle swarm optimization using digital pheromones, Struct. Multidiscip. Optim. 37 (2009) 415–427.
[11] A. Engelbrecht, F. van den Bergh, A cooperative approach to particle swarm optimization, IEEE Trans. Issue Evol. Comput. 8 (2004) 225–239.
[12] M.A. Potter, K.A.D. Jong, A cooperative coevolutionary approach to function optimization, III Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: Parallel Problem Solving from Nature (1994) 249–257.
[13] Y. Jiang, T. Hu, C. Huang, X. Wu, An improved particle swarm optimization algorithm, Appl. Math. Comput. 193 (2007) 231–239.
[14] M. Waintraub, R. Schirru, C.M. Pereira, Multiprocessor modeling of parallel particle swarm optimization applied to nuclear engineering problems, Prog. Nucl. Energy 51 (2009) 680–688.
[15] E. Alba, J.M. Troya, Analyzing synchronous and asynchronous parallel distributed genetic algorithms, Future Gen. Comput. Syst. 17 (2001) 451–465.
[16] E. Alba, F. Luna, A.J. Nebro, J.M. Troya, Parallel heterogeneous genetic algorithms for continuous optimization, Parallel Comput. 30 (2004) 699–719.
[17] J. Schutte, J. Reinbolt, B. Fregly, R. Haftka, A. George, Parallel global optimization with the particle swarm algorithm, Int. J. Numer. Meth. Eng. 61 (2004) 2296–2315.
[18] J.A. Reinbolt, J.F. Schutte, B.J. Fregly, B.I. Koh, R.T. Haftka, A.D. George, K.H. Mitchell, Determination of patient-specific multi-joint kinematic models through two-level optimization, J. Biomech. 38 (2005) 621–626.
[19] B.-I. Koh, A.D. George, R.T. Haftka, B.J. Fregly, Parallel asynchronous particle swarm optimization, Int. J. Numer. Meth. Eng. 67 (2006) 578–595.
[20] G. Venter, J. Sobieszczanski-Sobieski, A parallel particle swarm optimization algorithm accelerated by asynchronous evaluations, in: 6th World Congresses of Structural and Multidisciplinary Optimization, Rio de Janeiro, Brazil, pp. 1–10.
[21] G. Venter, J. Sobieszczanski-Sobieski, Multidisciplinary optimization of a transport aircraft wing using particle swarm optimization, Struct. Multidiscip. Optim. 26 (2004) 121–131.
[22] V.K. Kalivarapu, J.-L. Foo, E.H. Winer, Synchronous parallelization of particle swarm optimization with digital pheromones, Adv. Eng. Softw. 40 (2009) 975–985.
[23] V.K. Kalivarapu, E.H. Winer, Asynchronous parallelization of particle swarm optimization through digital pheromone sharing, Struct. Multidiscip. Optim. 39 (2009) 263–281.
[24] Y. Zheng, S. Chen, H. Ling, X. Xu, Multi-agent based distributed computing framework for master–slave particle swarms, J. Softw. 23 (2012) 3000–3008.
[25] K. Parsopoulos, Parallel cooperative micro-particle swarm optimization: a master–slave model, Appl. Soft Comput. J. 12 (2012) 3552–3579.
[26] Y. Chen, Y. Feng, X. Li, A parallel system for adaptive optics based on parallel mutation PSO algorithm, Optik 125 (2014) 329–332.
[27] J. Park, B.-I. Kim, A modified particle swarm optimization algorithm: information diffusion PSO, J. Korean Inst. Ind. Eng. 37 (2011) 163–170.
[28] B. Li, K. Wada, Communication latency tolerant parallel algorithm for particle swarm optimization, Parallel Comput. 37 (2011) 1–10.
[29] M. Gardner, A. McNabb, K. Seppi, A speculative approach to parallelization in particle swarm optimization, Swarm Intell. 6 (2012) 77–116.
[30] A.W. McNabb, C.K. Monson, K.D. Seppi, Parallel PSO using mapreduce, in: IEEE Congress on Evolutionary Computation (CEC 2007), IEEE Press, Piscataway, 2007, pp. 7–14.
[31] P. Subbaraj, R. Rengaraj, S. Salivahanan, T. Senthilkumar, Parallel particle swarm optimization with modified stochastic acceleration factors for solving large scale economic dispatch problem, Int. J. Elect. Power Energy Syst. 32 (2010) 1014–1023.
[32] K.-Y. Tu, Z.-C. Liang, Parallel computation models of particle swarm optimization implemented by multiple threads, Expert Syst. Appl. 38 (2011) 5858–5866.
[33] A. Farmahini-Farahani, S. Vakili, S.M. Fakhraie, S. Safari, C. Lucas, Parallel scalable hardware implementation of asynchronous discrete particle swarm optimization, Eng. Appl. Artif. Intell. 23 (2010).

[34] L. Mussi, F. Daolio, S. Cagnoni, Evaluation of parallel particle swarm optimization algorithms within the CUDA architecture, Inf. Sci. 181 (2011) 4642–4657.
[35] Y. Hung, W. Wang, Accelerating parallel particle swarm optimization via GPU, Optim. Meth. Softw. 27 (2012) 33–51.
[36] V. Roberge, M. Tarbouchi, Comparison of parallel particle swarm optimizers for graphical processing units and multicore processors, Int. J. Comput. Intell. Appl. 12 (2013) 1–20.
[37] A. Husselmann, K. Hawick, Levy flights for particle swarm optimisation algorithms on graphical processing units, Parallel Cloud Comput. 2 (2013) 32–40.
[38] A.O.S. Moraes, Desenvolvimento e implementação de versões paralelas do algoritmo do enxame de partículas em clusters utilizando MPI, 2011, Dissertação de mestrado Brasil, PEQ/COPPE/UFRJ, Rio de Janeiro, RJ.
[39] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J. Dongarra, MPI. The Complete Reference, The MPI Core, second ed., vol. 1, MIT Press, Cambridge, MA, USA, 1998.
[40] S. Surjanovic, D. Bingham, Virtual library of simulation and experiments. Test Functions and Datasets, 04 Feb. 2014. <http://www.sfu.ca/ssurjano/index.html>, 2014.
[41] J.F. Mitre, P.L.C. Lage, M.A. Souza, E. Silva, L.F. Barca, A.O.S. Moraes, R.C.C. Coutinho, E.F. Fonseca, Droplet breakage and coalescence models for the flow of water-in-oil emulsions through a valve-like element, Chem. Eng. Res. Des. 92 (2014) 2493–2508.
[42] D. Ramkrishna, Population Balances – Theory and Applications to Particulate Systems in Engineering, Academic Press, New York, 2000.
[43] P.L.C. Lage, R.C. Rodrigues, J.F. Mitre, M.N. d. Souza, L.F. Barca, M.A. Souza, E. Silva, S. Varella, R.C.C. Coutinho, Droplet evolution in the flow of water in oil emulsions through duct accidents – analysis of droplet breakage, in: Proceedings of the CFDOIL, pp. 1–41.
[44] J.F.M. Araujo, Modelos de quebra e coalescência de gotas para o escoamento de emulsões, Tese de D. Sc Brasil, PEQ/COPPE/UFRJ, Rio de Janeiro, RJ, 2010.
[45] J.W. Zwokak, P.T. Boggs, L.T. Watson, ODRPACK95, Technical Report, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, USA, 2004.