ISSN 2176-5480

22nd International Congress of Mechanical Engineering (COBEM 2013)
November 3-7, 2013, Ribeirão Preto, SP, Brazil

ASYNCHRONOUS PARALLELIZATION OF THE PARTICLE SWARM OPTIMIZATION ALGORITHM AND ITS APPLICATION TO PARAMETER ESTIMATION IN A LARGE DIMENSIONAL SPACE

Antonio de Oliveira Samel Moraes
Paulo Laranjeira da Cunha Lage
Argimiro Resende Secchi
Programa de Engenharia Química - Laboratório de Termofluidodinâmica (LTFD/PEQ/COPPE/UFRJ), Rio de Janeiro Brasil
amoraes@peq.coppe.ufrj.br, paulo@peq.coppe.ufrj.br, arge@peq.coppe.ufrj.br

Abstract. *Despite its recognized strength in global optimization problems, the high number of objective function evaluations requested by the Particle Swarm Optimization (PSO) method severely limits its application when the computational cost of the objective function calculation is large, as in most actual engineering problems. In order to overcome this deficiency to enable the efficient use of the PSO algorithm in large-scale engineering problems, the parallel computation in clusters appears as an excellent resource. Following this premise, an asynchronous parallelization of the PSO algorithm was developed in this work using the master-slave parallel paradigm and functions of the MPI (Message Passing Interface) library. A set of benchmark tests was conducted to validate and to analyze the performance of the developed method. The developed algorithm, named AIU-PPSO (Asynchronous and Immediate Update Parallel PSO), showed excellent performance, with linear speedup and parallel efficiency higher than 90% for all tested problems. The results were obtained using MIMD parallel computers with distributed memory and 20 Gbits/s Infiniband network. 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.*

Keywords: *Optimization, PSO, MPI, Asynchronous Parallelization, Parameter Estimation*

1. Introduction

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

The gradient-based methods, like the Newton and Conjugate Gradient ones, although effective, are fundamentally local, which makes them highly dependent on the initial conditions. In this context, global (deterministic or not) algorithms are highlighted and have been the subject of intense study nowadays. Examples of deterministic methods are the BARON, Branch and Reduce Optimization Navigator (Sahinidis and Tawarmalani, 2011) and the DIRECT, DIvinding RECTangles (Bjorkman and Holmstrom, 1999). On the other hand, examples of non deterministic are the *Genetic Algorithms*, GAs (Holland, 1992), the Simulated Annealing, SA (Metropolis *et al.*, 1953), the Adaptive Random Search, ARS (Secchi and Perlingeiro, 1989) and the Particle Swarm Optimization, PSO (Eberhart and Kennedy, 1995).

Particularly, non deterministic algorithms based on the behavior of populations (PBM - *Population Based Methods*) as GA and PSO, are highlighted due their recognized strength. The main deficiency of these methods is the high number of objective function evaluation required in the search process. For application in large-scale engineering problems, in which the calculation of 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. By its nature, these methods are easily parallelizable, which makes the development of their parallel strategies an open field of studies. Particularly, in the PSO algorithm, the calculation of the objective function is performed independently for each particle in the swarm and, therefore, can be efficiently parallelized.

In the present work, an asynchronous parallel version of the PSO, named *Asynchronous and Immediate Update Parallel PSO* (AIU-PPSO), using the MPI[1] (Message Passing Interface) library and the master-slave paradigm, was developed. This algorithm showed excellent performance, with linear speedup and parallel efficiency higher than 90% for all tested benchmarks problems. The results were obtained using MIMD parallel computers with distributed memory and 20 Gbits/s Infiniband network. The cluster nodes had dual Quad-Core AMD 2356 processors with 16 GB of memory. 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.

[1] OpenMPI distribution, versions 1.2.6 and 1.4.2.

ISSN 2176-5480

2. Particle Swarm Optimization

The PSO algorithm is a non deterministic global optimization method inspired by gregarious behavior found among several species in nature. Initially proposed by Eberhart and Kennedy (1995), it belongs to the class of the algorithms based on swarm behavior, which assumes that the behavior of each individual strongly depends on a social component. This social component is related to the ability of interaction among the individuals, through mechanisms of information exchange that can be implicit or explicit.

In the algorithm, each individual particle corresponds to a potential solution of the optimization problem. In other words, each individual is at a position in the n-dimensional space formed by the Cartesian product of the optimization variables, in which it moves with its own velocity. Starting from an initial population, the algorithm consists of updating the velocity and the position of each particle according to its own movement and to the best position 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 the particle on its own movement and the *social* term brings about the influence of the best swarm position in the movement of each particle. Equations 1 and 2 show the algorithm updating particles velocities and positions as proposed by Eberhart and Kennedy (1995).

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

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

In the above equations, $\mathbf{x}_i$ e $\mathbf{v}_i$ are, respectively, the position and the velocity of the particle i, $\mathbf{x}_{m,i}$ is the coordinate of 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 k index is the counter of flights (or iterations) of the algorithm. The c_1 and c_2 are the cognitive and social parameters, while r_1 and r_2 are two numbers drawn randomly in the $[0, 1]$ interval for each particle at each flight of the algorithm.

The first important variant of PSO was proposed by Shi and Eberhart (1998), which consisted in the introduction of the *inertia weight*, a multiplicative parameter of the first term on the right side of the Eq. 1. As this term considers the influence of the particle velocity on its own motion, the new parameter controls the balance between the local and global characteristics of the search. As observed by the authors, high values of the inertia weight make the swarm more exploratory, while small values favor the local search and increase the convergence speed. Equation 3 shows the velocity update of the particles with the inertia weight.

$$\mathbf{v}_i^{k+1} = w\mathbf{v}_i^k + c_1 r_1[\mathbf{x}_{m,i}^k - \mathbf{x}_i^k] + c_2 r_2[\mathbf{x}_g^k - \mathbf{x}_i^k] \tag{3}$$

Clerc (1999) proposed to use of a constriction factor (K), shown in Eq. 4, in order to ensure the convergence of the particles. Shi and Eberhart (1998) observed that the constriction factor does not change the basic form of Eq. 3, but only makes a redefinition of the parameters. The advantage is the auto-adjustment of this parameter value, as shown in Eqs. 5 and 6, which can ensure the stability of the swarm and the convergence of the particles.

$$\mathbf{v}_i^{k+1} = K[\mathbf{v}_i^k + c_1 r_1(\mathbf{x}_{m,i}^k - \mathbf{x}_i^k) + c_2 r_2(\mathbf{x}_g^k - \mathbf{x}_i^k)] \tag{4}$$

$$K = \frac{2}{\left| 2 - \varphi - \sqrt{\varphi^2 - 4\varphi} \right|} \tag{5}$$

$$\varphi = c_1 + c_2 \qquad \text{and} \qquad \varphi > 4 \tag{6}$$

Biscaia *et al.* (2004) 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. The fundamental nature of the algorithm is preserved, but the motion of the particles becomes different. The main advantage of this formulation is its unconditional stability for inertia weight values below one ($w < 1$), which is a larger region when compared with the standard PSO (van den Bergh and Engelbrecht, 2006). Equations 7, 8 and 9, adapted from Biscaia *et al.* (2004), show the new expressions of the particles position and velocity updates.

$$\mathbf{x}_i^{k+1} = \mathbf{X}_i^k + \exp\left(\frac{w-1}{2}\right)\left\{(\mathbf{x}_i^k - \mathbf{X}_i^k)\cos(\theta_k) + \left[\frac{1-w}{2}(\mathbf{x}_i^k - \mathbf{X}_i^k) + \mathbf{v}_i^k\right]\frac{\sin(\theta_k)}{\theta_k}\right\} \tag{7}$$

$$\mathbf{v}_i^{k+1} = \exp\left(\frac{w-1}{2}\right)\left\{\mathbf{v}_i^k\cos(\theta_k) - \left[(c_1 r_1 + c_2 r_2)(\mathbf{x}_i^k - \mathbf{X}_i^k) + \frac{1-w}{2}\mathbf{v}_i^k\right]\frac{\sin(\theta_k)}{\theta_k}\right\} \tag{8}$$

where

$$\theta_k = \sqrt{(c_1 r_1 + c_2 r_2) - \frac{(1-w)^2}{4}} \qquad \text{and} \qquad \mathbf{X}_i^k = \frac{c_1 r_1 \mathbf{x}_{m,i}^k + c_2 r_r \mathbf{x}_g^k}{c_1 r_1 + c_2 r_2} \tag{9}$$

ISSN 2176-5480

The PSO method has been consolidated as a good choice for global optimization problems, mainly due to its notorious robustness and apparent superiority relative to other PBMs. In this context, Hassan *et al.* (2004) compared PSO and AGs methods. Their results showed that the two 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 PBMs 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.

Chen and Zhao (2009) proposed an adaptation of the swarm size based on the diversity concept. This concept is defined through metrics on the arrangement of the particles in the search space. The work is based on the idea that near to the end of the search process, a big swarm is no longer necessary, because the diversity is small. On the other hand, when the diversity is big, a larger population gives a better chance to find the global optimum.

Kalivarapu *et al.* (2009) 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. The action of pheromones is introduced in the swarm by the last term of Eq. 10.

$$\mathbf{v}_i^{k+1} = w\mathbf{v}_i^k + c_1 r_1[\mathbf{x}_{m,i}^k - \mathbf{x}_i^k] + c_2 r_2[\mathbf{x}_g^k - \mathbf{x}_i^k] + c_3 r_3[\mathbf{P}_i^k - \mathbf{x}_i^k] \tag{10}$$

where r_3 is a random number drawn in the interval $[0, 1]$, c_3 is the confidence parameter in the pheromone and $\mathbf{P}_i^k$ is the particle target pheromone in the k-th iteration (or flight) of the algorithm.

Engelbrecht and van den Bergh (2004) used the cooperative learning concept, an idea taken from Potter and Jong (1994) that have implemented this concept in their version of AG. Instead of solving a simple AG, Potter and Jong (1994) 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.* (2007) showed an improved PSO (IPSO) algorithm, where the swarm is partitioned in sub-swarms, which can run independently the PSO 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.* (2009) in their neighbor island model.

The PSO, or its variants, has also been widely used in real engineering problems. Among examples which may be mentioned are Reinbolt *et al.* (2005), Meneses *et al.* (2009) and Patel and Rao (2010).

2.1 Parallel Algorithms of PSO

Although parallel versions of some optimization algorithms such as genetic algorithms, simulated annealing and gradient-based algorithms are quite popular, the PSO parallelization is relatively recent and still little explored in the literature. According to Waintraub *et al.* (2009), the genetic algorithms are the PBM like methods most exploited in parallel computation. Among works which may be mentioned are Alba and Troya (2001) and Alba *et al.* (2004).

The first parallelization of the PSO was developed by Schutte *et al.* (2004), 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 slaves processors. The parallel efficiency of their algorithm was tested in the optimization of a bio-mechanic system defined in previous work (Reinbolt *et al.*, 2005).

As studied by Koh *et al.* (2006), 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 the first 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. Armed with this feature, a FIFO (First In First Out) queue is created for arrangement of the particles, which will be sent from master to slaves and receive by master from slaves.

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.* (2009) developed a synchronous strategy for a PSO with digital pheromones.

Waintraub *et al.* (2009) used the notion of cooperative swarms to develop a neighbor island model. In this model, an island topology is created that allows exchange of information and the migration of particles between the islands. Each island develops an independent population of particles that can execute any PSO variants and parallel strategies. From generation to generation (iterations of the algorithm) the best particles of each island migrate to their neighbors carrying their informations. The results of this work shows a significant improvement of the parallel algorithm efficiency over the simple master-slave strategy.

ISSN 2176-5480

3. PSO Codes

In the codes implemented in the present work, we used the particle velocity and position updates given by Shi and Eberhart (1998) show in Eqs. 2 and 3. The default values of the user defined cognition and social parameters c_1 and c_2 are 1.5. The inertia weight is updated in a decreasing manner according to Eq. 11.

$$w = w_0 + (w_f - w_0)\frac{N_v}{N_{max} + N_v} \tag{11}$$

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 the permanence number counter. This last parameter is zero at the beginning of the search, and it is incremented by one at the end of a flight if the permanence criterion, defined by:

$$|f_g^k - f_g^{k-1}| < \epsilon_a + \epsilon_r |f_g^k| \tag{12}$$

is satisfied. In Eq. 12, 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 for the permanence criterion.

The stop criterion is defined by the displacement of the weighted average position of the swarm ($\overline{\mathbf{y}}$) between two flights with satisfied permanence criterion, according to Eq. 13.

$$\|\overline{\mathbf{y}}^{N_v} - \overline{\mathbf{y}}^{N_{v-1}}\| < \epsilon_a \tag{13}$$

where $\overline{\mathbf{y}}^{N_{v-1}}$ is the weighted average position of the swarm referring to the last evaluation of the stop criterion before the current. The weighted average position of the swarm is given by:

$$\overline{\mathbf{y}} = \sum_{i=1;i\neq g}^{N_p} \overline{\omega}_i \mathbf{y}_i \tag{14}$$

where N_p is the number of the particle in the swarm, $\overline{\omega}_i$ is the weight, defined by Eq. 15.

$$\overline{\omega}_i = \frac{\frac{1}{\|\mathbf{y}_i - \mathbf{y}_g\|}}{\sum_{i=1;i\neq g}^{N_p} \frac{1}{\|\mathbf{y}_i - \mathbf{y}_g\|}} \tag{15}$$

In the above equations, $\mathbf{y}_i = [\mathbf{x}_i | \tilde{f}_{m,i}]$ and $\mathbf{y}_g = [\mathbf{x}_g | \tilde{f}_g]$ are the vectors defined in the $\Re^{n+1}$ space, that include the normalized optimization variable coordinates ($\mathbf{x}_i$ and $\mathbf{x}_g$, normalized in the $[0, 1]$ interval) and the normalized objective function ($\tilde{f}_{m,i}$, $\tilde{f}_g$), concatenated at the end of the vector. The normalization of the objective function values is given by:

$$\tilde{f}_{m,i} = \frac{f_{m,i} - \min_i(f_{m,i}^0)}{\max_i(f_{m,i}^0) - \min_i(f_{m,i}^0)} \tag{16}$$

where $\min_i(f_{m,i}^0)$ and $\max_i(f_{m,i}^0)$ are the minimum and maximum values of the objective function between the particles at the initial population of the swarm.

All the norms used in the above equations are the normalized Euclidian norm, defined Eq. 17:

$$\|\mathbf{x}\| = \left[\frac{1}{n}\sum_{i=1}^{n} x_i^2\right]^{\frac{1}{2}} \tag{17}$$

3.1 Serial Code

The implementation of serial PSO algorithms is very simple. After initialization of the positions and velocities of the particles (normally in an aleatory manner), the particles go to a loop (flight loop) in which their velocities and positions are updated according to the best values of objective function calculated for each particle and for the swarm. At the end of each flight, a permanence criterion, given by Eq. 12 is tested. When this test is consecutively satisfied N_{min} (the user defined minimum number of optimum permanence) times, a stop criterion is tested. Algorithm 1 shows the pseudo-code of the serial PSO algorithm. In this pseudo-code, Max_{aval} is the user defined maximum number of objective function evaluations, the $rand()$ function give a aleatory number in the range $[0, 1]$ and N_p is the number of particles.

ISSN 2176-5480

Algorithm 1 *Pseudo code of the serial implementation of the PSO algorithm.*

Require: $w_0, w_f, c_1, c_2, N_p, N_{min}, N_{max}$ e 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
  $(f_g, j) \leftarrow min_i(f_{m,i}, i)$
  $\mathbf{x}_g \leftarrow \mathbf{x}_j$
  $k \leftarrow 0$
  $N \leftarrow 0$
  $N_v \leftarrow 0$
  $optimum \leftarrow f_g$
  $N_{aval} \leftarrow N_p$
  while $(N < N_{max}$ and $N_{aval} < Max_{aval})$ **do**
    Update w with Eq. 11
    for $(i = 1 \rightarrow N_p)$ **do**
      Update $\mathbf{x}_i$ and $\mathbf{v}_i$ (equations 2 and 3)
      $N_{aval} \leftarrow N_{aval} + 1$
      if $(F_{obj}(\mathbf{x}_i) < f_{m,i})$ **then**
        $f_{m,i} \leftarrow F_{obj}(\mathbf{x}_i)$
        $\mathbf{x}_{m,i} \leftarrow \mathbf{x}_i$
      end if
      if $(F_{obj}(\mathbf{x}_i) < f_g)$ **then**
        $f_g \leftarrow F_{obj}(\mathbf{x}_i)$
        $\mathbf{x}_g \leftarrow \mathbf{x}_i$
      end if
    end for
    Performs permanence and stop criteria
    $k \leftarrow k + 1$
  end while

Permanence and stop criteria
  if (permanence criterion) **then**
    $N \leftarrow N + 1$
    $N_v \leftarrow N_v + 1$
    if $\left(\frac{N}{N_{min}} \in \aleph\right)$ **then**
      if (stop criterion) **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

3.2 Parallel Codes

The parallel codes were developed using a master-slave strategy for data communication between processors. In this strategy, the master processor coordinates all steps of the code and manages all the communications with the slaves processors, which calculate the objective function.

3.2.1 Parallel synchronous code (IU-PPSO)

The parallel synchronous code of PSO developed in the present work, named *Immediate Update Parallel PSO* (IU-PPSO), can be seen as a simple extension of the serial code. It is essentially identical to the serial code, with the difference that the objective function is performed in parallel mode. The *point-to-point MPI_Send()* and *MPI_Recv()* communication functions of MPI library (Snir *et al.*, 1998) were used for data communication between processors. Algorithm 2 shows the pseudo-code of the algorithm performed by the master processor. The slaves only perform the objective function calculation, as showed by the Algorithm 3.

3.2.2 Parallel asynchronous code (AIU-PPSO)

The synchronous algorithm shows a weakness on its parallel strategy: *The need of synchronization makes the code limited by the slower slave*. On the other words, in the synchronization point (at the end of the flight loop), the master must wait that all slaves finish their work to continue the algorithm having any available slave in an idle waiting state. According to Schutte *et al.* (2004), this algorithm performs better when (i) it executes in a homogeneous cluster, (ii) the computational time of the objective function is homogeneous and (iii) the tasks can be equally distributed between the slaves processors. The failure of any of these conditions tends to cause unbalanced load, degrading the performance of the algorithm.

In order to overcome this limitation, an asynchronous strategy of parallelization was implemented, in which *there is no synchronization point*. Thus, the flight concept is destroyed, rising in its place the pseudo-flight concept. The pseudo-flight consists of a chosen number of the objective function evaluations after which the master processor exits the optimization step to test the convergence criterion. However, all tasks being made by the master are carried out without stopping the work that is being done by the slaves. So, the only time loss in the parallelization occurs when a slave

ISSN 2176-5480

Algorithm 2 *Pseudo code of the algorithm performed by master processor in the IU-PPSO code.*

Require: $w_0, w_f, c_1, c_2, N_p, N_{proc}, N_{min}, N_{max}, Max_{aval}$
  $N_{slaves} = N_{proc} - 1$
  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
  $(f_g, j) \leftarrow min_i(f_{m,i}, i)$
  $\mathbf{x}_g \leftarrow \mathbf{x}_j$
  $k \leftarrow 0$
  $N \leftarrow 0$
  $N_v \leftarrow 0$
  $optimum \leftarrow f_g$
  $N_{aval} \leftarrow N_p$
  Performs Optimization step
  Optimization step (flight loop)
  while $(N < N_{max}$ e $N_{aval} < Max_{aval})$ **do**
    Update w with Eq. 11
    for $(i = 1 \rightarrow N_p)$ **do**
      if $(i < N_{slave})$ **then**
        Update $\mathbf{x}_i$ and $\mathbf{v}_i$ (equations 2 and 3)
        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 $\mathbf{v}_i$ (equations 2 and 3)
        Send $\mathbf{x}_i$ to slave p
      end if
    end for
    synchronizes the slaves
    Performs permanence and stop criteria (see Algorithm 1)
  end while

Algorithm 3 *Pseudo code of the step performed by slaves.*

  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

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. Thus, the slaves may perform different number of function evaluation, which make the asynchronous code not to maintain the consistency described in Schutte *et al.* (2004). The Algorithm 4 shows the pseudo-code of the optimization step (pseudo-flight loop) performed by the master processor in the AIU-PPSO.

3.3 Evaluation Metrics

The parallel algorithms were evaluated using a set of benchmark problems. The speedup and parallel efficiency concepts are used to this evaluation. The speedup (fair speedup, S) is defined as the ratio between the computational time spent by the serial code and that one spent by parallel codes:

$$S = \frac{T_s}{T_p} \tag{18}$$

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

The fair speedup shows the effective gain that the parallel code 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 showed by Eq. 19.

$$\eta = \frac{S}{N_{proc}} \tag{19}$$

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. 18 is the mean computational time of the N_{run} runs. As

ISSN 2176-5480

Algorithm 4 *Pseudo code of the optimization step performed by master processor in the AIU-PPSO code.*

while ($N < N_{max}$ and $N_{aval} < Max_{aval}$) **do**
  Update w with Eq. 11
  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**
    $f_g \leftarrow F_{obj}$
    $\mathbf{x}_g \leftarrow \mathbf{x}_j$
  end if
  Remove from the queue the next particle (i) to be calculated
  Update $\mathbf{x}_i$ and $\mathbf{v}_i$ (equations 2 and 3)
  Send $\mathbf{x}_i$ to slave p
  if (End of pseudo-flight) **then**
    Performs permanence and stop criteria such as in Algorithm 1
  end if
end while

each numerical experiment is independent, a relation to calculate the standard deviation of the speedup is necessary. This relation is provided by Eq. 20, whose demonstration is provided in appendix A of Moraes (2011).

$$\sigma_S^2 = \frac{[\sigma_{T_s}^2 + E(T_s)^2][\sigma_{T_p}^2 + E(T_p)^2]}{E(T_p)^4} - \left[\frac{E(T_s)}{E(T_p)}\right]^2 + \frac{\sigma_{T_p}^2}{E(T_p)^4}\left[2\sigma_{T_s}^2 - \sigma_{T_p}^2 \frac{E(T_s)^2}{E(T_p)^2}\right] \tag{20}$$

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

4. Benchmark test functions

Several benchmark test functions were used to evaluate the algorithms. In this work, the results are presented for multidimensional Ackley, Schwefel and H1 functions, defined by the equations 21, 22 and 23, respectively.

$$F(\mathbf{x}) = -20\exp\left(-0,2\sqrt{\sum_{i=1}^{n}\frac{x_i^2}{n}}\right) - \exp\left(\sum_{i=1}^{n}\frac{\cos 2\pi x_i}{n}\right) + 20 + \exp(1); \quad \mathbf{x} \in [-32.68, 32.68]^n \tag{21}$$

where the global minimum is at the origin,

$$F(\mathbf{x}) = 418n - \sum_{i=1}^{n} x_i sen[\sqrt{(|x_i|)}]; \quad \mathbf{x} \in [-500, 500]^n \tag{22}$$

where the global minimum is at $\mathbf{x} = [420.9687, 420.9687]^n$ and

$$F(\mathbf{x}) = \frac{sen^2\left[x_1 - \frac{x_2}{8}\right] + sen^2\left[x_2 + \frac{x_1}{8}\right]}{1 + \sqrt{(x_1 - 8,6998)^2 + (x_2 - 6,7665)^2}}; \quad \mathbf{x} \in [-100, 100]^n \tag{23}$$

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

Other results can be found in Moraes (2011).

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 4. Table 1 shows the user defined parameters used for these three problems. In order to emulate a larger problem, a random delay was added to the evaluation of the objective function. Table 2 shows the resulting computational time for Schwefel function with two optimization variables and for Ackley function with thirty-two (32) variables.

ISSN 2176-5480

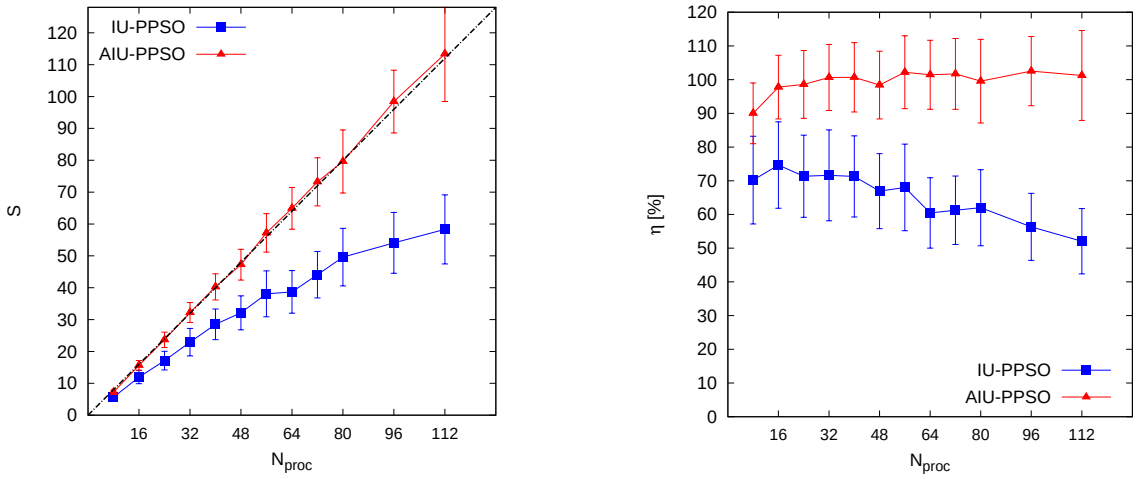
Figures 1 and 2 show the speedup (S) and efficiency (η) for the synchronous (IU-PPSO) and asynchronous (AIU-PPSO) algorithms in the optimization of Schwefel 2D and Ackley 32 D functions, respectively. The vertical bars in the graphs correspond to two standard deviations up and down of the mean, calculated for 100 independently numerical experiments using different initial guesses.

Table 1: User defined parameters used for the two tested benchmark problems.

Test function	Np	c_1	c_2	w_0	w_f	N_{min}	N_{max}	Max_{aval}	ϵ_a and ϵ_r
Schwefel (2D)	150	1.5	1.5	1.0	0.1	20	80	10^7	10^{-4}
Ackley (32D)	1000	1.5	1.5	1.0	0.1	40	160	10^7	10^{-4}
H1	50	1.5	1.5	1.0	0.1	20	80	10^7	10^{-4}

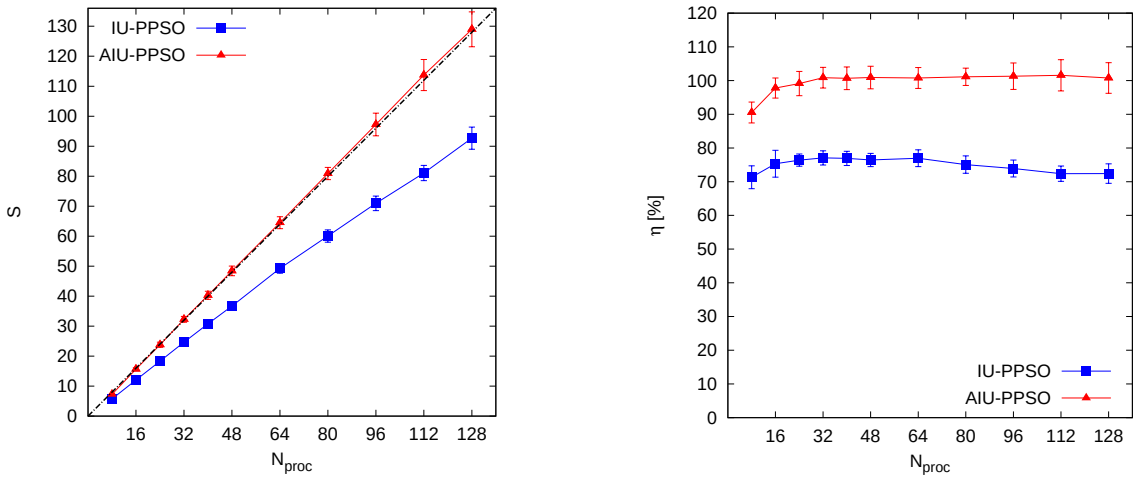
Table 2: Computational costs of the objetive fuction for the Schwefel 2D and Ackley 32D test functions.

Test function	CPU Time [s]
Schwefel (2D)	0.02537 ± 0.006893
Ackley (32D)	0.03179 ± 0.005794



(a) (b)

Figure 1: Speedup and efficiency of the parallel algorithms for Schwefel 2D test function: (a) the speedup and (b) the efficiency.



(a) (b)

Figure 2: Speedup and efficiency of the parallel algorithms for Ackley 32D test function: (a) the speedup and (b) the efficiency.

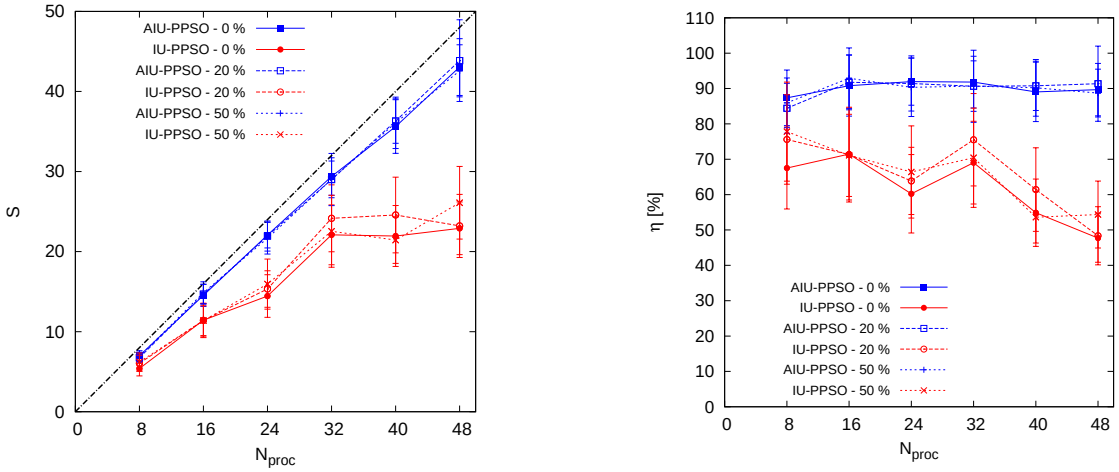
ISSN 2176-5480

The asynchronous algorithm shows linear speedup with mean parallel efficiency above 90% for the two test functions. On the other hand, the performance of the synchronous algorithm degrades when the number of processors increases. The higher efficiency of the AIU-PPSO is due to the nature of the parallelization. As commented in the Section 3.2.2, the asynchrony of the code 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 small. On the other hand, the synchronization point in the IU-PPSO makes this loss intrinsic.

Table 3 shows the computational time for H1 function with maximum variability degrees of 0, 20 and 50%, as was done by Koh *et al.* (2006). In other words, the CPU time is randomly distributed within the ranges listed in the Tab. 3. Figure 3 shows the speedup and the efficiency of the IU-PPSO and AIU-PPSO in the optimization of these problems. Again, the AIU-PPSO code 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.* (2006), the CPU time variability of the objective function did not affect the performance of the parallel algorithms developed in the present work.

Table 3: Computational costs of the objetive fuction for the H1 2D function.

	Mean CPU Time [s]	CPU Time range [s]
Null variability	0.5	[0.5, 0.5]
Maximum variability of 20 %	0.5	[0.5, 0.6]
Maximum variability of 50 %	0.5	[0.5, 0.75]



(a) (b)

Figure 3: Speedup and efficiency of the parallel codes for H1 test function: (a) the speedup and (b) the efficiency.

5.2 Parameter estimation problem

The AIU-PPSO was applied to parameter estimation of a *Population Balance Model* for emulsion flows with droplet breakage and coalescence (Araújo, 2010). This model describe the evolution of the droplets numerical density distribution function whose moments provide physical properties of the droplets in the flow, such as diameter, interface area and internal energy. Details of the model are described by Araújo (2010), who also developed the models for droplets breakage and coalescence, whose parameters are estimated here.

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

$$y_i \cong f(x_i; \beta) \tag{24}$$

where i is the experiment indice, y_i and x_i are, respectively, the values of the response and explanatory variables, both experimentally obtained, $f(x_i; \beta)$ are the values of the response variables according to the model and β are the model parameters.

Assuming a perfect model, the relative error (ϵ_i) of the model prediction for the i-th experiment is:

$$\epsilon_i = f_i(x_i + \delta_i; \beta) - y_i \tag{25}$$

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

$$\delta_i = \overline{x}_i - x_i \tag{26}$$

ISSN 2176-5480

where $\overline{x}_i$ 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 estimation parameter problem, as used by the ODRPACK95 optimizer (Zwokak *et al.*, 2004), is given by :

$$F_{obj} = \sum_{i=1}^{N_{exp}} \left[\epsilon_i^T W_{\epsilon_i} \epsilon_i + \delta_i^T W_{\delta_i} \delta_i\right] \tag{27}$$

where N_{exp} is the number of experiments, W_{ϵ_i} is the weight of the each response variable and W_{δ_i} is the weight of each explanatory variable.

The CPU time of the objective function given by Eq. 27 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 (β_1, β_2 and β_3). The mean CPU time was 1138.93 s with a standard deviation of 187.65 s. For this problem, 78 experiments were considered. Since there is another parameter whose value depends on the flow configuration for each experiment, this enlarges the dimension of the parameter space to 81. The default values of the user defined parameters ($w_0 = 1$, $w_f = 0.1$ and $c_1 = c_2 = 1.5$) of the AIU-PPSO were used. Furthermore, 2000 particles were employed and the tolerance used in the stop criteria was 10^{-3}.

Figure 4 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 330119 objective function evaluations, taking 66 days to be completed. The application of serial codes of PSO or other PBMs would be unfeasible, since the computational cost of evaluating the objective function 330119 times, would take about 12 years in the same processor cores.

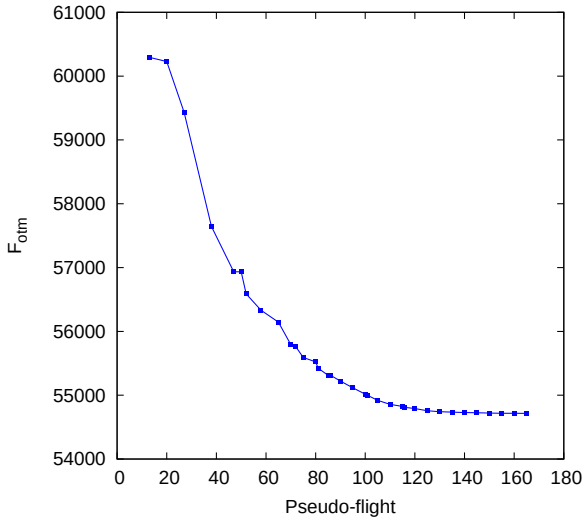


Figure 4: Evolution of the objective function value along the pseudo-flights of the optimization process.

6. Conclusion

For all cases of parallel scalability analysis, the AIU-PPSO provided the best performance, with linear speedup and mean parallel efficiency above 90%, even when using all 128 processors of the LTFD 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.

For the estimation parameter problem, the AIU-PPSO made possible to use the PSO algorithm in an engineering problem with 81 parameters. Due the computational time of the objective function, it is impractical to use serial PSO or other PBM to solve it.

7. ACKNOWLEDGEMENTS

Antonio de O. S. Moraes acknowledges the financial support from CNPq, grants no. 140951/2012-1. Paulo L. C. Lage acknowledges the financial support from CNPq, grants nos. 302963/2011-1 and 478589/2011-5, and FAPERJ, grant no. E-26/111.361/2010. Argimiro R. Secchi acknowledges the financial support from CNPq, grants 304907/2009-0 and 480040/2010-9.

ISSN 2176-5480

8. REFERENCES

Alba, E., Luna, F., Nebro, A.J. and Troya, J.M., 2004. "Parallel heterogeneous genetic algorithms for continuous optimization". *Parallel Computing*, Vol. 30, No. 5-6, pp. 699–719. Parallel and nature-inspired computational paradigms and applications.

Alba, E. and Troya, J.M., 2001. "Analyzing synchronous and asynchronous parallel distributed genetic algorithms". *Future Generation Computer Systems*, Vol. 17, No. 4, pp. 451–465.

Araújo, J.F.M., 2010. *Modelos de quebra e coalescência de gotas para o escoamento de emulsões*. Tese de D.Sc., PEQ/COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

Biscaia, J.E.C., Schwaab, M. and Pinto, J.C., 2004. "Um novo enfoque do método do enxame de partículas". *Proceedings of the Workshop em Nanotecnologia e Computação Inspirada na Biologia*.

Bjorkman, M. and Holmstrom, K., 1999. "Global optimization using the direct algorithm in matlab". *AMO. Advanced Modeling and Optimization*, Vol. 1, No. 2.

Chen, D. and Zhao, C., 2009. "Particle swarm optimization with adaptive population size and its application". *Applied Soft Computing*, Vol. 9, No. 1, pp. 39–48.

Clerc, M., 1999. "The swarm and the queen: toward a deterministic and adaptative particle swarm optimization". *Evolutionary Computation. CEQ 99*, pp. 1951–1957.

Eberhart, R. and Kennedy, J., 1995. "Particle swarm optimization". *Proceedings IEEE International Conference on Neural Networks*, pp. 1942–1948.

Engelbrecht, A. and van den Bergh, F., 2004. "A cooperative approach to particle swarm optimization". *Evolutionary Computation, IEEE Transactions on Issue*, Vol. 8, No. 2, pp. 225–239.

Hassan, R., Cohanim, B., Weck, O. and Venter, G., 2004. "A comparison of particle swarm optimization and the genetic algorithm". Technical report, Massachusetts Institute of Technology, Cambridge, MA, 02139.

Holland, J.H., 1992. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press Cambridge, MA, USA, 1st edition. ISBN 0-262-08213-6.

Jiang, Y., Hu, T., Huang, C. and Wu, X., 2007. "An improved particle swarm optimization algorithm". *Applied Mathematics and Computation*, Vol. 193, No. 1, pp. 231–239.

Kalivarapu, V., Foo, J.L. and Winer, E., 2009. "Synchronous parallelization of particle swarm optimization with digital pheromones". *Advances in Engineering Software*, Vol. 40, No. 10, pp. 975–985.

Koh, B.I., George, A.D., Haftka, R.T. and Fregly, B.J., 2006. "Parallel asynchronous particle swarm optimization". *International Journal for Numerical Methods in Engineering*, Vol. 67, No. 4, pp. 578–595.

Meneses, A.A.M., Machado, M.D. and Schirru, R., 2009. "Particle swarm optimization applied to the nuclear reload problem of a pressurized water reactor". *Progress in Nuclear Energy*, Vol. 51, No. 2, pp. 319–326.

Metropolis, N., Rosenbluth, A.W., Teller, M. and Teller, E., 1953. "Equation of state calculations by fast computing machines". *The Journal of Chemical Physics*, Vol. 21, No. 6, pp. 1087–1092.

Moraes, A.O.S., 2011. *Desenvolvimento e implementação de versões paralelas do algoritmo do enxame de partículas em clusters utilizando MPI*. Dissertação de mestrado, PEQ/COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

Patel, V. and Rao, R., 2010. "Design optimization of shell-and-tube heat exchanger using particle swarm optimization technique". *Applied Thermal Engineering*, Vol. 30, No. 11-12, pp. 1417–1425.

Potter, M.A. and Jong, K.A.D., 1994. "A cooperative coevolucionary 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*, pp. 249–257.

Reinbolt, J.A., Schutte, J.F., Fregly, B.J., Koh, B.I., Haftka, R.T., George, A.D. and Mitchell, K.H., 2005. "Determination of patient-specific multi-joint kinematic models through two-level optimization". *Journal of Biomechanics*, Vol. 38, No. 3, pp. 621–626.

Sahinidis, N. and Tawarmalani, M., 2011. *BARON*. Carnegie Mellon University, Department of Chemical Engineering, Pittsburgh, USA.

Schutte, J., Reinbolt, J., Fregly, B., Haftka, R. and George, A., 2004. "Parallel global optimization with the particle swarm algorithm". *International Journal for Numerical Methods in Engineering*, Vol. 61, No. 13, pp. 2296–2315.

Secchi, A.R. and Perlingeiro, C.A., 1989. "Busca aleatória adaptativa". In *Proceedings of XII Congresso Nacional de Matematica Aplicada e Computacional*. São José do Rio Preto, SP, pp. 49–52.

Shi, Y. and Eberhart, R.C., 1998. "Parameter selection in particle swarm optimization". *In Evolutionary programming VII: Proceedings of the EP98, New York: Springer-Verlag*.

Snir, M., Otto, S., Huss-Lederman, S., Walker, D. and Dongarra, J., 1998. *MPI. The Complete Reference: Volume 1, The MPI Core*. MIT Press Cambridge, MA, USA, 2nd edition. ISBN 0-262-69215-5.

van den Bergh, F. and Engelbrecht, A., 2006. "A study of particle swarm optimization particle trajectories". *Information Sciences*, Vol. 176, No. 8, pp. 937–971.

Waintraub, M., Schirru, R. and Pereira, C.M., 2009. "Multiprocessor modeling of parallel particle swarm optimization

ISSN 2176-5480

applied to nuclear engineering problems". *Progress in Nuclear Energy*, Vol. 51, No. 6-7, pp. 680–688.

Zwokak, J.W., Boggs, P.T. and Watson, L.T., 2004. "Odrpack95". Technical report, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, USA.

9. RESPONSIBILITY NOTICE

The authors are the only responsible for the printed material included in this paper.