Computers and Chemical Engineering 81 (2015) 344–354

Contents lists available at ScienceDirect

Computers and Chemical Engineering

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

Parallel optimization by means of a Spectral-Projected-Gradient approach

Juan Ignacio Ardenghi[b], Gustavo Esteban Vazquez[b,c], Nélida Beatriz Brignole[a,b,*]

[a] Planta Piloto de Ingeniería Química (PLAPIQUI) Complejo CCT-UAT, CONICET, Camino La Carrindanga Km. 7, 8000 Bahía Blanca, Argentina
[b] Laboratorio de Investigación y Desarrollo en Computación Científica (LIDeCC), Departamento de Ciencias e Ingeniería de la Computación (DCIC), Universidad Nacional del Sur (UNS), Av. Alem 1253, Bahía Blanca, Argentina
[c] Universidad Católica del Uruguay, Facultad de Ingeniería y Tecnologías, Av. 8 de Octubre 2801, Montevideo, CP 11600, Uruguay

ARTICLE INFO

Article history:
Received 31 October 2014
Received in revised form 2 April 2015
Accepted 8 April 2015
Available online 17 April 2015

Keywords:
Optimization
Parallel programming
Non-linear problems
Efficiency

ABSTRACT

The judicious exploitation of the inherent optimization capabilities of the Spectral-Projected-Gradient method (SPG) is proposed. SPG was implemented in order to achieve efficiency. The novel adjustments of the standard SPG algorithm showed that the parallel approach proves to be useful for optimization problems related to process systems engineering. Efficiency was achieved without having to relax the problems because the original model solutions were obtained in reasonable time.

1. Introduction

It is widely recognized that parallelizing is far from trivial. Parallel models that run in multiple processors are highly modified versions of the corresponding sequential solvers. In fact, parallel algorithms constitute new techniques with their own advantages and drawbacks. Though they can succeed in being faster, their main design difficulties are sometimes related to communication management, interaction between runs, memory requirements or experimental evaluation (Alba, 2005).

El-Rewini and Lewis (1997) had also pointed out that parallel programming involves all the difficulties that comprehend serial programming, together with additional challenges, such as data or task partitioning, parallel debugging, and synchronization. Unlike single-processors systems, interconnection bandwidth and message latency dominate the performance of parallel systems. Moreover, there is no evident way to predict the performance of a new system. Therefore, prior to a significant investment of time and effort, it is difficult to envisage clearly the benefits of parallelizing.

Parallel programming does involve several daunting challenges, but it is worthwhile! Problems not solved before become nowadays solvable by using parallel algorithms. According to Buzzi-Ferraris and Manenti (2010), parallel computing on personal computers is taking its steps as a silent revolution that directly involves many other scientific and industrial areas, naturally including Process Systems Engineering (PSE) and Computer-Aided Process Engineering (CAPE) communities.

In the search of more realistic formulations, the need for more rigorous modelling, but together with the modern global requirement of faster solutions, has grown. Hence, cost-effective solutions are required in order to be able to address effectively large-scale problems, which have proved to be very demanding in terms of computational effort and efficacy, i.e. the length of time devoted to problem-solving.

These days parallel programming has turned into an attractive field that deserves to be carefully exploited. It is growing fast, with an enormous application potential. Then, it constitutes a promising methodology needing more attention by the PSE optimization community. How to address time-consuming problems is undoubtedly a subject of interest and concern for chemical engineers, who have sometimes resorted to parallelism. For instance, Abdel-Jabbar et al. (1998) implemented a partially decentralized state observer on multicomputers demonstrating the potential of parallel processing in the field of model-based control. In turn, Chen et al. (2011) have also accelerated their molecular weight distribution (MWD)

* Corresponding author at: Planta Piloto de Ingeniería Química (PLAPIQUI) Complejo CCT-UAT, CONICET, Camino La Carrindanga Km. 7, 8000 Bahía Blanca, Argentina. Tel.: +54 291 4861700.
E-mail address: dybrigno@criba.edu.ar (N.B. Brignole).

http://dx.doi.org/10.1016/j.compchemeng.2015.04.010
0098-1354/© 2015 Elsevier Ltd. All rights reserved.

calculation method by means of parallel programming. Cheimarios et al. (2013) also exploited parallelism when modelling a chemical vapor deposition (CVD) reactor. The time-consuming computations in the micro-scale were efficiently accelerated thanks to the implementation of a synchronous master-worker parallel technique. In turn, Laird et al. (2011) have addressed large-scale dynamic optimization problems with a decomposition approach helpful to exploit parallel computing for the Karush–Kuhn–Tucker (KKT) system. Lately, for the contingency-constrained alternating current optimal power flow (ACOPF) problem Kang et al. (2014) have reported the achievement of significant improvements in solution times by means of their parallel Schur-complement based, nonlinear interior-point method.

Improving both the convergence and solution time of process system optimization problems is nowadays of great significance. The community would greatly benefit by deepening HPC knowledge across the chemical engineering field, especially research involving industry-standard software development and implementation (Piccione, 2014).

Broadly speaking, for the past two decades many non-linear optimization problems in the PSE area have been solved by posing them with a wide variety of strategies, such as parallel processing, model reformulation, model decompositions, convergence-depth control and surrogate-based approaches. Efficient non-traditional algorithmic alternatives have been proposed in order to reduce the computational cost of solving some demanding problems. Kheawhom (2010) reported a constraint handling scheme that exhibited a considerably lower computational cost than the cost required by the traditional penalty function. In turn, Kraemer et al. (2009) proposed a reformulation for complex large-scale distillation processes that required significantly less computational time in order to identify local optima of better quality. In process design and control, Wang et al. (2007) showed advantageous numerical results by using convergence depth control. For process control, Abdel-Jabbar et al. (1998) designed a parallel algorithm that guaranteed stability and optimal performance of the parallel observer. Later, for the problem of integrated design and control optimization of process plants, Egea et al. (2007) proposed surrogate-based methods that compete with conventional control strategies.

Besides, for optimization problems related to planning issues, and always working from a sequential point of view – i.e. without exploiting any opportunity of parallelism – You et al. (2011) aimed at the reduction of computational time by means of model reformulation. With a view to solving large-scale instances effectively, they proposed the following computational strategies: (I) a two-level solution strategy and (II) a continuous approximation method. Their approaches led to the same optimal solutions, but with different CPU times. Moreover, for their problem about the simultaneous route selection and tank sizing approach, You et al. (2011) pointed out that solving the aggregated model may become intractable as the problem size increases, due to the combinatorial complexity of route enumeration.

In contrast, the PSE problems are sometimes solved by relaxing variables and conditions, thus generating non-linear subproblems easier to tackle through linear and quadratic approximations. An important and interesting question is the following: Can we achieve efficiency without having to reformulate nor to relax the problem?

Nowadays, it seems natural that the trust-region methods occupy a significant place in the PSE simulation area. Nevertheless, Spectral Projected Gradient (SPG) methods constitute an inspiration for the acceleration of optimization algorithms via parallelization. SPG (Birgin et al., 2000) was born from the merging of the Barzilai–Borwein (spectral) non-monotone concepts with classical projected gradient strategies (Bertsekas, 1976). Some authors, like Raydan (1997) and Fletcher (2005), analyzed this kind of methods carefully.

2. Solving optimization problems by taking advantage of parallel processing

We can rely on parallel computing in order to reduce computing times significantly, without being necessary to resort to strategies that imply model simplifications or problem reformulations. It is efficient and practical, though it is a daunting challenge to program it carefully. Moreover, parallel computing is useful to complement other approaches. In this work parallel programming has been applied to enhance an algorithm originally proposed by Birgin et al. (2000), who developed a method that was born as a combination of spectral nonmonotone ideas (Grippo et al., 1986) with classical projected gradient strategies (Barzilai and Borwein, 1988).

2.1. The mathematical problem

The optimization problem considered here is a non-linear programming (NLP) problem. Its model involves a non-linear objective function $f(x)$ subjected to a set of equality constraints $c_i(x) = 0$, $i = 1, \ldots, n_i$; a set of inequality constraints, $c_j(x) \geq 0$, $j = 1, \ldots, n_j$; and upper and lower bounds on the continuous variables x_i. Any of the functions involved in both kinds of constraints can be non-linear. In its algebraic form, the general problem is given by Eq. (1)

$$\begin{cases} & \min_x f(x) \\ s.t. & c_i(x) = 0, \quad i = 1, \ldots, n_i \\ & c_j(x) \geq 0, \quad j = 1, \ldots, n_j \\ & l_i \leq x_i \leq u_i \end{cases} \tag{1}$$

By introducing the slack variables z_j in the inequality constraints, Eq. (1) turns into Eq. (2).

$$\begin{cases} & \min_x f(x) \\ s.t. & c_i(x) = 0, \quad i = 1, \ldots, n_i \\ & c_j(x) - z_j = 0, \quad j = 1, \ldots, n_j \\ & l_i \leq x_i \leq u_i \\ & z_j \geq 0 \end{cases} \tag{2}$$

Both the objective function and the equality constraints can be combined into an augmented Lagrangian function (Eq. (3) and Eq. (4)), where $\Lambda \in \mathbb{R}^{n_i+n_j}$ is an estimate of the vector of Lagrange multipliers, $\rho > 0$ is the penalty parameter and $\|\cdot\|$ is the Euclidean norm.

$$\mathcal{L}(x, \lambda, \rho) = f(x) + C(x)^T \Lambda + \left(\frac{\rho}{2}\right) \cdot \left\|C(x)\right\|^2 \tag{3}$$

$$C(x) = \{c_i(x)\} \cup \{c_j(x) - z_j\} \tag{4}$$

Eq. (2) is reformulated to Eq. (5) by means of Eq. (3). The problem stated in Eq. (5) becomes box-constrained because the sole explicit constraints are the variable bounds, while the rest of the constraints are embedded in the Augmented Lagrangian.

$$\begin{cases} & \min_x \mathcal{L}(x, \Lambda, \rho) \\ s.t. & l_i \leq x_i \leq u_i \\ & z_j \geq 0 \end{cases} \tag{5}$$

This change makes it easier to compute the projections onto the feasible region; thus, the general algorithmic performance is improved. This property represents a great advantage for a low-cost algorithm, like the spectral projected gradient method. By virtue of Eq. (5), we shall henceforth refer to the Lagrangian as the objective function of the optimization problem.

2.2. The Spectral Projected Gradient approach

In chemical engineering the most widely used methods for optimization are those derived from Newton's method, which all share the same property: they are descent methods. In contrast, the Spectral Projected Gradient (SPG) approach gives way to a different class of optimization methods, where there is no requirement that the function decreases alongside each iteration. The spectral step length is the solid foundation on which SPG is built. When gradient directions are coupled with suitable spectral step lengths, they produce really better results than the traditional Cauchy approach, often being competitive with the performance of the Newton or quasi-Newton methods (Gomes-Ruggiero et al., 2009). Moreover, SPG is quite attractive due to its low computational cost. SPG neither computes Hessian matrices nor solves linear systems; it just uses projections onto the feasible region, matrix-vector products and a non-monotone line search.

The projected gradient method is an efficient algorithm to solve bound constrained optimization problems (Nocedal and Wright, 1999). The Spectral Projected Gradient approach (SPG) gives way to a class of low-cost optimization algorithms structured by merging the projected gradient method with two key features of optimization methods: the implementation of the spectral step length introduced by Barzilai and Borwein (1988) and a non-monotone strategy for function minimization developed by Grippo et al. (1986).

Raydan (1997) proved its global convergence and exhibited numerical experiments that showed its efficiency. SPG algorithms have gained importance because of their relatively little requirement of computational work and satisfactory global convergence behaviour. SPG has successfully been applied to constrained problems in various fields like geophysics (Cores et al., 2000), physics (Birgin et al., 1999), chemistry (Wells et al., 1994), Process Systems Engineering (Domancich et al., 2004) and control theory (Ardenghi et al., 2008). Moreover, SPG is also one of the main constituents of algorithms to solve non-linear equation systems (La-Cruz and Raydan, 2003), partial differential equations (Molina and Raydan, 1996) and other non-linear programming problems (Luengo et al., 2002). Besides, SPG has been merged with other algorithms for different optimization problems (Birgin et al., 2000). In particular, Crema et al. (2007) presented and discussed encouraging numerical results for their combination of the subgradient method with the spectral choice of step-length and a computational cost per iteration. Later, Júdice et al. (2008) reported an efficient variant of SPG combined with a specially designed line search in order to find a solution to the symmetric eigenvalue complementarity problem.

The framework of an SPG algorithm (Fig. 1) is built with routines to compute the projections onto the feasible set, the objective function and its gradient. The general steps are described below.

Algorithm SPG: ***Spectral Projected Gradient***

```
Step 1. Check for convergence
Step 2. Compute direction and step length
Step 3. Compute a new iterate
Step 4. Call non-monotone Line Search procedure
Step 5. Go to step 1
```

Fig. 1. The structure of a general SPG algorithm.

- *Step 1*: is implemented either by checking through Eq. (6) whether either the norm of the projection is lower than a specified tolerance tol_1.

$$\left\|\mathrm{Proj}(x_k - \nabla\mathcal{L}(x_k)) - x_k\right\| < \mathrm{tol}_1 \tag{6}$$

or when a stationary point is detected by means of Eq. (7)

$$\left\|\nabla\mathcal{L}(x_k)\right\| < \mathrm{tol}_2 \cdot (1 + |\mathcal{L}(x_k)|) \tag{7}$$

- *Step 2*: If $g_k = \nabla\mathcal{L}(x_k)$ is denoted, the gradient of $\mathcal{L}$ at point x_k, direction d_k is given by Eq. (8).

$$d_k = -g_k \tag{8}$$

and the spectral step length is given by Eq. (9), where $s_k = x_{k+1} - x_k$ and $y_{k-1} = g_k - g_{k-1}$

$$\lambda_k = \frac{s_{k-1}^T s_{k-1}}{s_{k-1}^T y_{k-1}}. \tag{9}$$

- *Step 3*: The candidate for the new iterate is given by Eq. (10).

$$x_{k+1} = x_k + \lambda_k d_k. \tag{10}$$

- *Step 4*: The procedure for the non-monotone line search is the following:

  Given a sufficient decrease parameter $\gamma \in (0, 1)$, a chosen value $t \in (0, 1)$, and $\mathcal{L}_{\max}$, which is the maximum value of the Lagrangian among the last M iterations,

```
While  $\mathcal{L}(x_k + 1) > \mathcal{L}_{\max} + t\gamma\lambda_k d_k^T g_k$,  do
    $x_{k+1} = x_k + t\lambda_k d_k$
End while
```

- *Step 5*: $k = k + 1$
  Go to Step 1.

As to γ, it is a sufficient-decrease parameter, which is fixed and defined by the user. γ regulates the variation of Lagrangian's values. For Step 4, t is employed to reduce the step length in the while loop. The sequential version systematically reduces the length of the search space selecting either $t = ½$ or an intermediate point by means of a quadratic interpolation (Birgin et al., 2001). In our implementation t is a parameter whose value depend on the number of processors p. This reduction is performed through different simultaneous values of t, i.e. $t_i = i/(p+1)$, $i = 1, \ldots, p$. Hence, $i/(p+1) < 1$.

As to iterative behaviour, algorithmic convergence is based on the proximity of the spectral step length to an eigenvalue of the Hessian matrix evaluated in the optimum. Therefore, it is not required for the objective function to decrease from iteration to iteration, which is the property that characterizes SPG as non-descent (Birgin et al., 2000). A non-descent method requires a certain regulation in the objective function in order to be able to reach the global optimum. For SPG, the non-monotone line search is the key to guarantee the global convergence (Raydan, 1997).

It should be noted that, unlike many optimization procedures, information about the Hessian matrix is not required. For each of the iterations this algorithm only needs both the gradient and a non-monotone line-search strategy in order to ensure global convergence. Moreover, the number of algebraic operations is linear in terms of the dimension n of the optimization variable x_k (Birgin et al., 2009). These features make SPG suitable to solve the subproblem derived by the augmented Lagrangian (Eq. (5)). The Augmented Lagrangian combined with Spectral-Projected-Gradient (ALSPG) was implemented sequentially by Diniz-Ehrhardt et al. (2004) and its numerical performance proved to be very good when compared with sophisticated trust-region algorithms.

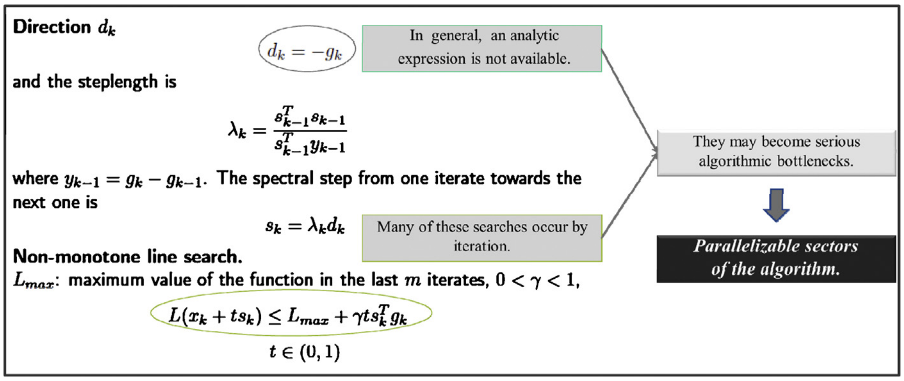

Direction d_k
$d_k = -g_k$
In general, an analytic expression is not available.
and the steplength is
$\lambda_k = \frac{s_{k-1}^T s_{k-1}}{s_{k-1}^T y_{k-1}}$
where $y_{k-1} = g_k - g_{k-1}$. The spectral step from one iterate towards the next one is
$s_k = \lambda_k d_k$
Many of these searches occur by iteration.
They may become serious algorithmic bottlenecks.
Parallelizable sectors of the algorithm.
Non-monotone line search.
L_{max}: maximum value of the function in the last m iterates, $0 < \gamma < 1$,
$L(x_k + ts_k) \leq L_{max} + \gamma t s_k^T g_k$
$t \in (0, 1)$

Fig. 2. Key time bottlenecks to be handled through parallellization.

On the other hand, the necessary number of inner iterations for iteration k of Step 4 depends on the objective function on a neighbourhood of point x_k. The ideal situation would be that the original step size (Eq. (9)) had been accepted. If not, that step size should be reduced as described in Step 4 of the SPG algorithm. This procedure does not impose a decrease in the objective function. In contrast, a parameter M is chosen at the beginning of the process, then a trial point is accepted when there is enough diminution, in comparison with the maximum function value among the last M iterations (see Step 4).

2.3. Parallel design for SPG

The low number of algebraic operations makes SPG's arithmetic cost strongly dependent on function evaluations, which are focused on both the gradient evaluation and the non-monotone line search. Therefore, this structural feature is inherently exploitable since these procedures are clearly parallelizable sectors. The possibility of performing these tasks in parallel gives us the opportunity to redesign SPG to make it highly efficient.

The need for an accurate approximation of the gradient in Step 2 and an eventually high number of inner iterations in Step 4 may become significant bottlenecks in the main algorithm. Nevertheless, both steps are parallelizable areas (Fig. 2). Therefore, a variant form is worthwhile to be implemented.

Let us consider a coarse-grain parallel version of ALSPG, i.e. without introducing parallelism in the arithmetic operations. The main goal is the development of an efficient version of the algorithm programmed in parallel, assuming lack of knowledge about the analytical expressions of the objective function or gradients.

The general scheme of the Augmented Lagrangian algorithm adopted here is the one developed by Hestenes (1969) and Powell (1969) with the practical implementation described by Nocedal and Wright (1999). In this approach the inner subproblem is solved by any suitable method chosen by the user. The novelty of our proposal is the inclusion in this general scheme a variant that also works in parallel.

The stage where our parallel version of SPG is incorporated to solve the inner subproblem is illustrated in Fig. 3. The flowchart shows the combination of the classical Augmented Lagrangian scheme with our parallel SPG solver. We have baptized this algorithm pALSPG (parallel Augmented Lagrangian with Spectral Projected Gradient).

Two steps were parallelized: gradient calculation and step length determination. For gradient calculation, one processor is employed for each dimension of vector x_k. Each processor receives vector x_k and the index of the coordinate it should calculate. The task-pool paradigm was implemented. So, when a processor finishes its work, another dimension of x_k is immediately allocated to it. As to step length determination, each processor is allocated the Lagrangian evaluation for each value of t_i.

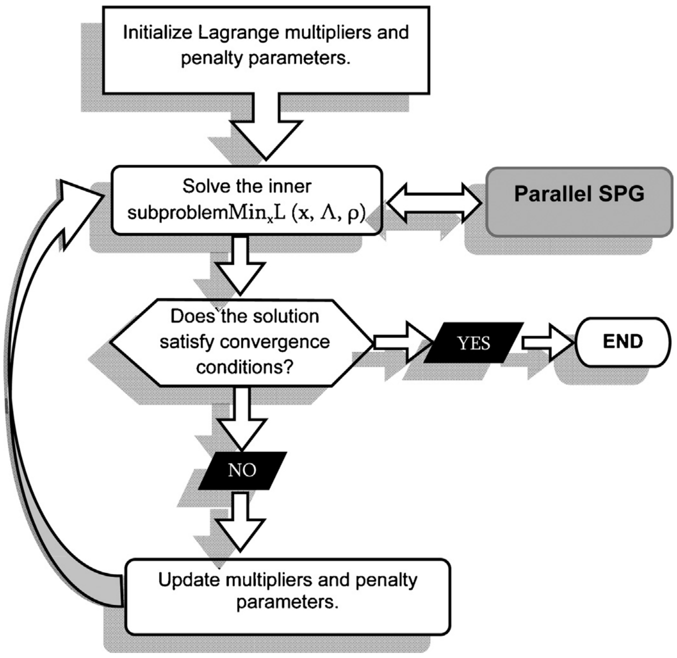

Initialize Lagrange multipliers and penalty parameters.
Solve the inner subproblemMin$_x$L (x, Λ, ρ)
Parallel SPG
Does the solution satisfy convergence conditions?
YES
END
NO
Update multipliers and penalty parameters.

Fig. 3. A simple flowchart representing the pALSPG algorithm.

Due to SPG's distribution of tasks, the step length determination is not carried out in parallel simultaneously when the gradient is evaluated. Then, both the line search and the gradient evaluation are performed with the whole set of available processors.

2.3.1. The gradient

When a clear expression of the objective function cannot be written in an analytical way the derivatives should be approximated by some alternate way. In this work we have employed the well-known central difference formula (Eq. (11)), where h_i is a small constant, e_i is the elementary vector in the ith direction, n is the dimension of vector x and the last term refers to the magnitude of the approximation error, which is proportional to h_i^2 (Dennis and Schnabel, 1983)

$$f'(x) = \frac{f(x + h_i e_i) - f(x - h_i e_i)}{2h_i} + \mathcal{O}(h_i^2), \quad i = 1, \ldots, n \tag{11}$$

It should be noticed that function evaluations within a derivative calculation are totally independent from each other. Therefore, it is possible to calculate them simultaneously. By using parallel processing, this entire process can be carried out efficiently.

One of our main design goals is to create a user-friendly environment, where the user does not necessarily provide analytic derivatives. For this purpose, numerical derivatives were chosen because they are easy to implement and straight-forward to parallelize.

2.3.2. The line search

In standard methods the objective function is usually enforced to decrease at every iteration in order to achieve global convergence. Even so, by forcing these conditions at successive iterations, convergence slows down. In SPG the gradient behaviour is associated with the step length. It is desired that the gradient vanishes. The spectral step length tries to approximate the eigenvalues of the Hessian matrix evaluated at the optimum. When the spectral length approaches an eigenvalue, some gradient components tend to zero, but the rest of its components are also affected. Eventually, they can grow together with the objective function. Gradient methods that either forbid non-monotonic steps or limit their effects are only able to remove some components slowly. Hence, they suffer from slow convergence (Fletcher, 2005). Therefore, in order to achieve a fast convergence, at successive iterations a non-monotone line search determines whether the step will be accepted.

As it was stated in Step 4 (Fig. 1), given $0<\gamma<1$ and $0<t<1$, the iterate x_{k+1} is accepted if

$$\mathcal{L}(x_{k+1}) \leq \mathcal{L}_{\max} + t\gamma d_k^T g_k \tag{12}$$

where $\mathcal{L}_{\max}$ is the maximum function value among the last M iterations.

In the non-monotone line search a backtracking takes place. The parallel version partitions the search space in p intervals of length $1/p$, where p is the number of available processors or threads. Then, at certain iterations the point accepted by the parallel line search procedure may differ from the one originally allowed by the sequential one, without implying a different final solution. Then, the partial results may not be the same, which does not necessarily mean that the solution is unlike.

The simultaneous evaluation of the objective function at these points allows us to determine a satisfactory point in a single call to the line-search procedure. In contrast, the sequential procedure has to resort to particular strategies, such as quadratic interpolation or any other backtracking resource, to find a satisfactory point (Raydan, 1997).

The rejection of too many trial points increases the number of inner iterations for the line search, which may dominate the general algorithmic performance. In consequence, very high computational times are consumed. Therefore, a parallel line search was designed.

Let us say that there are p processors available $\{P_1, P_2, \ldots, P_p\}$. At iterate k we take p points from the interval $[x_k, x_k+\lambda_k d_k]$ and evaluate the condition (Eq. (12)) simultaneously. Each thread has an index i, and the inequation (Eq. (12)) is evaluated in the point $x_k+(i/(p+1))\lambda_k d_k$. The guiding principle relies on dividing the work into separate units and distributing them to the processors. The processor scheduling is dynamic, allowing us to keep a better balance among operations, which cannot be accurately predicted.

In general terms our implementation follows the well-known parallel Master-Worker model (Gropp et al., 1994). The Master role was assigned to a central processor. His job is to distribute the tasks among the remaining processors (called the Workers), to collect their task results and finally combine them into an overall solution. As to the data partitioning, a coarse-grain parallel-distributed version was implemented. Instead of parallelizing the arithmetic operations, our version introduces parallelism both in the gradient approximation and in the line search procedure. The implicit objective was to minimize the communication and synchronization times.

The task partitioning is different depending on the algorithmic stage. On the one hand, the task distribution for the evaluation of the gradient vector is dynamic. A partial derivative is associated to an individual task that finds the corresponding gradient component. Parallelism arises naturally when dealing with gradient vector, since each vector component is an independent unit. Whenever each Worker has finished his computation, he sends his value to the Master who checks what component has not been calculated or assigned yet and sends him that pending task. This dynamic distribution is advantageous since it is fundamental to avoid the existence of idle processors in the system.

On the other hand, the line search is stationary because each Worker has a fixed parameter that allocates his task. Whenever the task is finished, either the answer or a failure signal is sent to the Master. He synchronizes the results and decides when either to continue with SPG or to wait for more answers. When the Master receives a point that satisfies condition (12) and he determines that there is no chance of receiving a point with a step size closer to the spectral length, he stops the search and orders to go on with the main SPG iteration. In this way, time is saved because the Master never waits for the appearance of better solutions.

2.3.3. Efficiency evaluation

A few concepts that are useful for efficiency measurements are presented in Bertsekas and Tsitsiklis (1997). The ratio S_p (Eq. (13)) is called the speed-up factor for a parallel system of size p (i.e. with p processors or p threads). In Eq. (13) T_s is the serial time required to solve the problem and T_p is the time taken by a parallel system of size p to solve the same problem.

$$S_p = \frac{T_s}{T_p} \tag{13}$$

For a given problem, the efficiency of an algorithm measures how the processors are exploited in order to solve the computational problem. The percentage of efficiency E_p (Eq. (14)) is the ratio between the speed-up S_p and the size of the parallel system p, multiplied by 100.

$$E_p = 100 \times \frac{S_p}{p} \tag{14}$$

When analyzing procedural efficiency, it is necessary to assess the time taken by the communications. For parallel systems, the "parallel time" spent with p processors, T_p, may be determined by using the approximate formula given by Eq. (15), where T_{Cpt} is the computing time, i.e. how long it takes for the multiprocessor system to make arithmetic operations, and T_{Com} is the communication time, i.e. the time the multiprocessor system takes in order to execute data transfers.

$$T_p \approx T_{Cpt} + T_{Com}. \tag{15}$$

The computing time for the parallel gradient evaluation T_{Cpt} is given by Eq. (16), where T_f is the exclusive computing time of the objective function and $r \in \mathbb{Z}$, $0<r<n$.

$$T_{Cpt} = \begin{cases} 2T_f \dfrac{n}{p} & \textit{if } n \bmod p = 0 \\ 2T_f \left(\dfrac{n-r}{p} + 1\right) & \textit{if } n \bmod p \neq 0 \end{cases} \tag{16}$$

In turn, the communication time can be set as $T_{Com} = \alpha + \beta l$, where l is the message length, α is the latency (i.e.the time required to start the message), $\beta = 1/\theta$ is the communication time and θ is the bandwidth.

For the parallel gradient, in the communication between the master process and the workers, the first p messages contain all data and for the next messages only the coordinate to be calculated is sent. Each worker returns the calculated derivative. Then, the communication time is given by Eq. (17)

$$T_{Com} = \underbrace{(\alpha+\beta n)p + (n-p)(\alpha+\beta)}_{Master \to workers} + \underbrace{(\alpha+\beta)(n+1)n}_{workers \to Master} \qquad (17)$$

Thus, the expected speed-up is given by Eq. (18)

$$S_p = \begin{cases} \dfrac{2nT_f}{2T_f\left(\dfrac{n-r}{p}+1\right) + (\alpha+\beta n)p + (n-p)(\alpha+\beta) + (\alpha+\beta)(n+1)n} & \text{in the worst case} \\ \dfrac{2nT_f}{2T_f\dfrac{n}{p} + (\alpha+\beta n)p + (n-p)(\alpha+\beta) + (\alpha+\beta)(n+1)n} & \text{in the best case} \end{cases} \qquad (18)$$

If the complexity order of T_f is higher than n^2 ($T_f \in \Omega(n^2)$), then $lim_{n\to\propto} S_p = p$. This represents an efficiency that tends to 100% when n is large.

For the line search, each worker receives from the master a packet with data coming from vector x_k, the gradient g_k, the steplength λ_k and an index i. Then, each one returns a success/failure message, the value of the function at x_k and its respective index.

Then, the communication time is given by Eq. (19)

$$T_{Com} = \underbrace{(\alpha+2\beta(n+1))p}_{Master \to workers} + \underbrace{2\beta p}_{workers \to Master} \qquad (19)$$

Worst case: It appears when the worker threads are deployed and the optimum point comes up first in the search, i.e. the point $x_k + (p/(p+1))\lambda_k d_k$ satisfies the line search condition (Eq. (12)). Then, the expected speed-up is given by Eq. (20).

$$S_p = \frac{T_f}{T_f + (\alpha+2\beta(n+1))p + 2\beta p} \qquad (20)$$

If $T_f = \Omega(np)$, then $lim_{n\to\propto} S_p = 1$. This represents an efficiency that tends to $1/p$ % when n is large, i.e. the efficiency decreases with the addition of processors.

Best case: It appears when the optimum point comes up last in the search, i.e. the point that satisfies the line search condition (Eq. (12)) is $x_k + (1/(p+1))\lambda_k d_k$. Then, the expected speed-up is given by Eq. (21).

$$S_p = \frac{pT_f}{T_f + (\alpha+2\beta(n+1))p + 2\beta p} \qquad (21)$$

If $T_f = \Omega(np)$, then $lim_{n\to\propto} S_p = p$. This represents an efficiency that tends to 100% when n is large.

2.4. Implementations

Two versions of the parallel algorithm were developed and implemented: one of them was conceived to run efficiently on a distributed architecture by using message passing on a standard local area network; the other one was designed to run on a shared local-memory architecture. The following parallel systems were employed:

- System I (distributed architecture): a cluster of 8 Pentium-4 processors interconnected by a standard local area network and PVM as the message-passing protocol.
- System II (multicore platform): 12 simultaneous worker threads in a Supermicro AS-4042G-TRF, compiled with OpenMP for multithreading programming.

3. Performance analysis

Birgin et al. (2000), when referring to the SPG method, stated that "It is quite surprising that such a simple tool can be competitive with rather elaborate algorithms which use extensively tested subroutines and numerical procedures".

The sequential SPG method has been tested with plenty of problems. In contrast, we have applied a parallel SPG strategy in several demanding problems of widespread interest in chemical engineering. This search strategy naturally differs from the sequential procedure, then it returns distinct results. There is a broad spectrum of applications that may benefit from this approach, including some inner problems belonging to the frameworks of widely comprehensive superstructures.

3.1. Case studies

The classical PSE problems that were selected for the tests are demanding, thus being necessary to solve them fast. The new parallel SPG implementation was evaluated by testing the following models:

- *Test Case 1*: The Synthesis of a Distillation-Based Separation System (TC1)
  This problem involves a three-component feed mixture that has to be separated into two multi-component products. The cost of each separator depends linearly on the flowrate through the separator, and the constraints correspond to mass balances around the various splitters, separators and mixers. This is a general mixed integer nonlinear programming (MINLP) problem, where this superstructure may involve the solution of NLP problems. We have taken Floudas's settings so as to run this test.
- *Test Case 2*: Propane, Isobutane, n-Butane, Non-sharp Separation (TC2)
  This test problem involves a three-component feed mixture that has to be separated into two three-component products. To avoid the distribution of non-key components the recoveries of the key components were set to be greater than 0.85. This problem has the same superstructure of Test Case 1, but the desired product composition is different.
- *Test Case 3*: Multicomponent Separation (TC3)
  In this problem a four-component mixture has to be separated into two multi-component products. The number of columns NC is 3. The three associated binary variables give rise to seven NLP problems.
- *Test Case 4*: Reactive Column (TC4)
  The model for a reactive distillation column consists in pure separation stages combined with a reactive distillation sector, where both reaction and separation take place at the same time.

The kind of problems that may benefit from parallel programming can be grouped under the title of 'unwieldy' problems because they consume significant computing time for various reasons, like those modelled with thermodynamic functions or many constraints. In particular, reactive distillation exhibits the typical behaviour of a computationally demanding problem.

The NLP models in TC1, TC2 and TC3 constitute various scenarios defined in the test collection by Floudas and Pardalos (1990), whose configurations come up as different parts of a big MINLP superstructure. For such a demanding framework the computational yield is fundamental because it is necessary to visit several NLPs to reach an optimum in the superstructure (Aggarwal and Floudas, 1990).

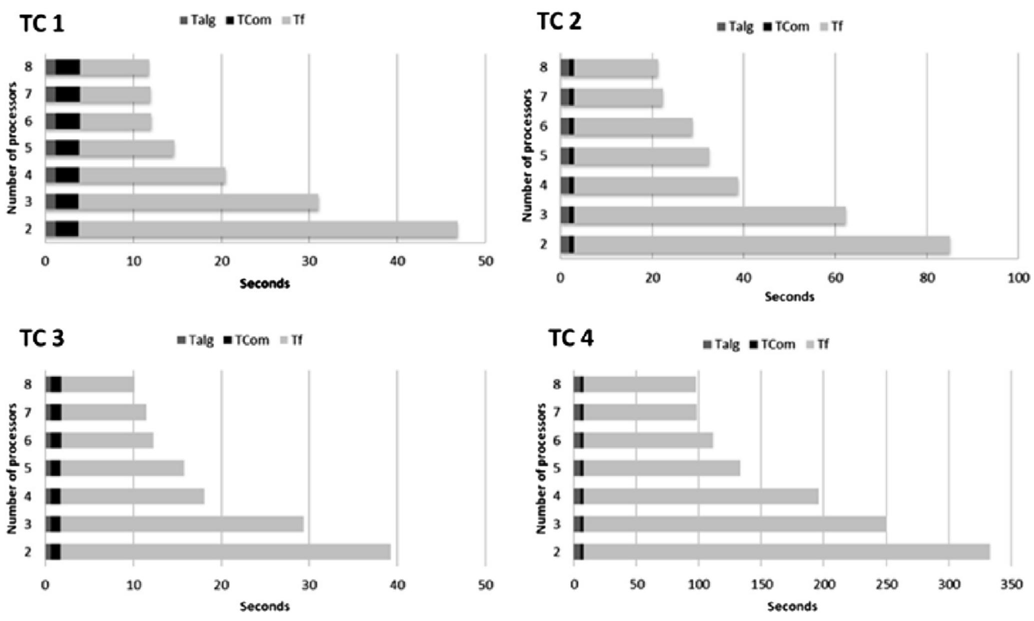


Fig. 4. Performance of System I for TC1–4.

As to TC4, the example chosen is the model of a reactive distillation column that yields methyl tert-butyl ether (MTBE) from a feed of methanol and isobutene (Domancich et al., 2009). For the base case, a rigorous equilibrium model is adopted and this column has 15 separation trays, 8 also being reactive stages. This problem is big and computationally demanding because 60% of all model equations include cumbersome thermodynamic equilibrium functions.

The Test Cases were numbered in ascending size order, i.e. according to the increasing amount of variables and equations that the adopted models contain. Table 1 reports the number of variables and constraints that were considered for each model.

3.2. Results and discussion

For the problems described above, the value of the objective functions and the norms of the constraints in the solution are summarized in Table 2.

Table 1
Size of the optimization problems for the testbed.

Test case	1	2	3	4
# of continuous variables	38	48	86	1173
# of linear constraints	17	13	22	227
# of nonlinear constraints	15	25	46	930

Table 2
Results for test problems TC1–4.

Test cases	TC1	TC2	TC3	TC4
$f(x)$	1.861	0.998	1.645	2244.7
$\|C(x)\|$	1.1×10^{-11}	1.2×10^{-8}	4.6×10^{-8}	2.0×10^{-2}

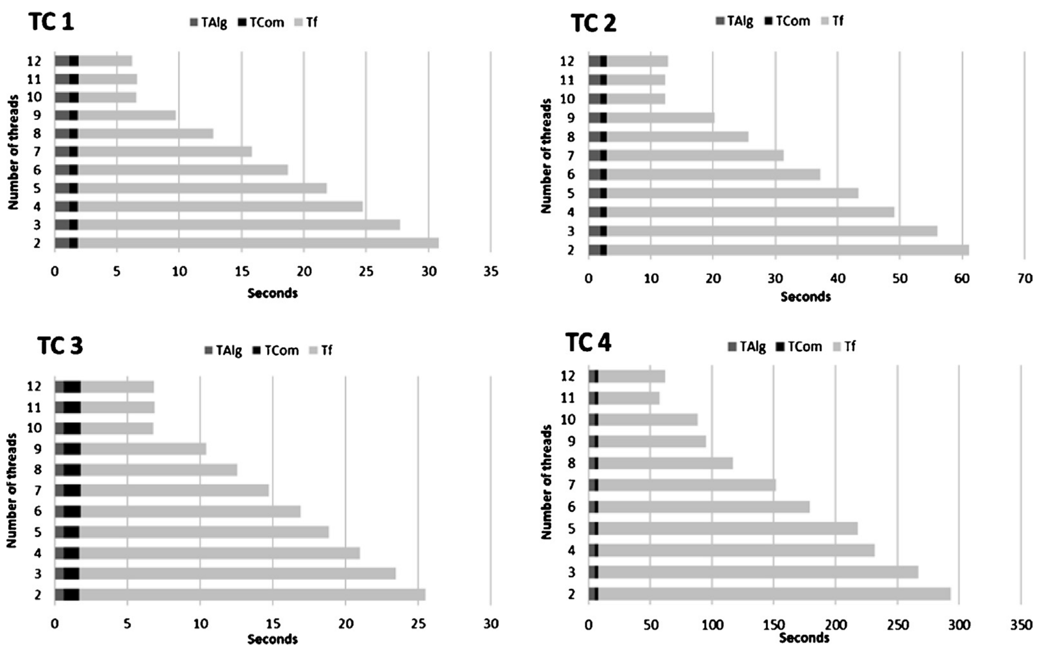


Fig. 5. Performance of System II for TC1–4.

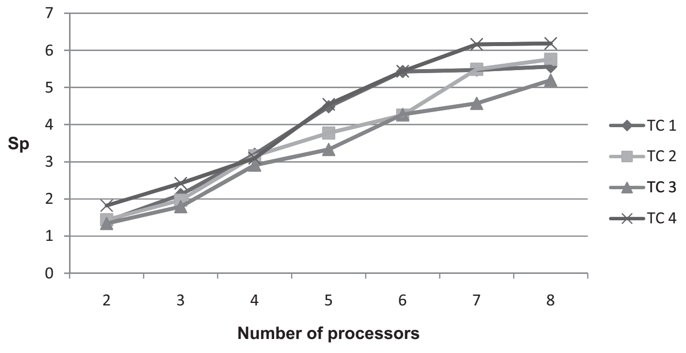


Fig. 6. Speed-up curves for TC1–4 on a cluster (System I).

Effectiveness and efficiency should be addressed in order to evaluate a process-systems algorithm. Effectiveness is the capacity of finding the solution of a given problem, while efficiency is the potential to find this solution in reasonably low run times, even for large-scale industrial problems. Since in Ardenghi et al. (2007) the effectiveness of pALSPG has been assessed for many problems, our interest is now focused on the reduction of the run time.

Figs. 4 and 5 show the time distribution between T_{Com} (in black) and T_{Cpt} (in grey). The computing time was split into the time exclusively taken by function evaluation T_f (in light grey) and the time spent in the remaining algebraic operations T_{Alg} (in dark grey). It can be observed that for both systems, increasing the number of threads slightly augments the communication time, but substantially reduces the time taken in function evaluations. On the other hand, there is an imperceptible change in T_f for 10–12 threads when using a multicore platform (Fig. 5). As a result, a plateau has ensued in Fig. 7.

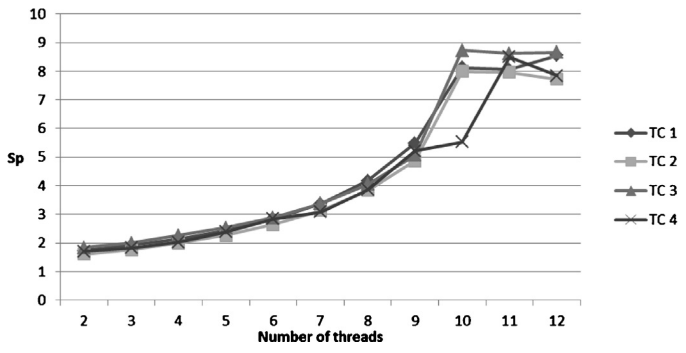


Fig. 7. Speed-up curves for TC1–4 on a multicore platform (System II).

Table 3
CPU times (s) for TC1–4 on a cluster (System I).

ALSPG				
TC1	TC2		TC3	TC4
65.74	122.46		52.58	606.87
No. processors	pALSPG			
	TC1	TC2	TC3	TC4
2	46.92	85.04	39.23	332.92
3	31.12	62.3	29.34	250.55
4	20.54	38.75	18.06	195.98
5	14.67	32.44	15.78	133.4
6	12.11	28.78	12.3	111.6
7	12.02	22.3	11.5	98.5
8	11.82	21.26	10.12	98.1

Table 4
CPU times (s) for TC1–4 on a multicore platform (System II).

ALSPG				
TC1	TC2		TC3	TC4
53.22	98.44		43.66	488.77
No. threads	pALSPG			
	TC1	TC2	TC3	TC4
2	30.82	61.1	23.81	283.05
3	27.73	55.98	21.75	254.65
4	24.74	49.12	19.3	245.51
5	21.82	43.31	17.18	204.55
6	18.71	37.25	15.11	183.43
7	15.83	31.37	12.93	155.06
8	12.74	25.71	10.77	117.02
9	9.71	20.2	8.62	92.2
10	6.55	12.32	5.01	88.39
11	6.61	12.36	5.06	57.56
12	6.22	12.74	5.04	62.37

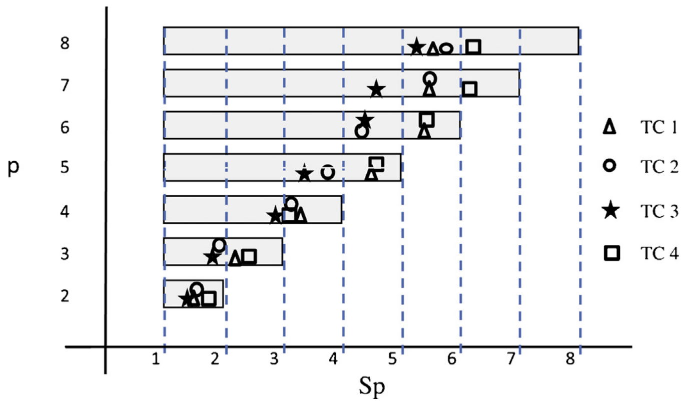


Fig. 8. An illustration of the computational behaviour of S_p for clusters (System I) on a given quantity of processors p.

3.2.1. Speed-up

For both implementations, Tables 3 and 4 show the actual CPU times (in seconds) for TC1–TC4. In both tables the third row corresponds to the elapsed time taken by the original serial algorithm when executed on a single computer of the corresponding system. These tables are useful to compare sequential and parallel times. The speed-up values illustrated in Figs. 6 and 7 were calculated with these experimental results. The number of iterations in the

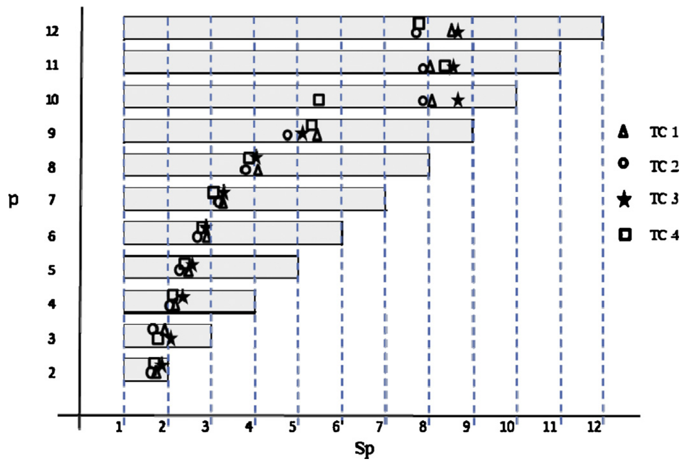
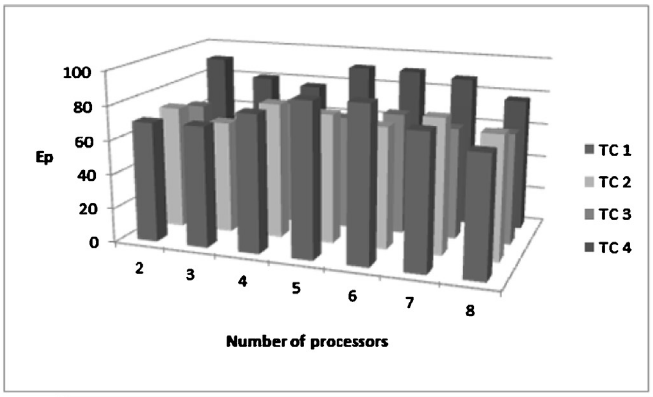


Fig. 9. An illustration of the computational behaviour of S_p for a multicore platform (System II) on a given quantity of processors p.



	Average	
N° proc.	Ep	Red_abs
2	75,05	32,43
3	69,1	51,18
4	77,3	67,62
5	80,69	74,8
6	80,82	79,07
7	77,48	81,35
8	70,95	82,31

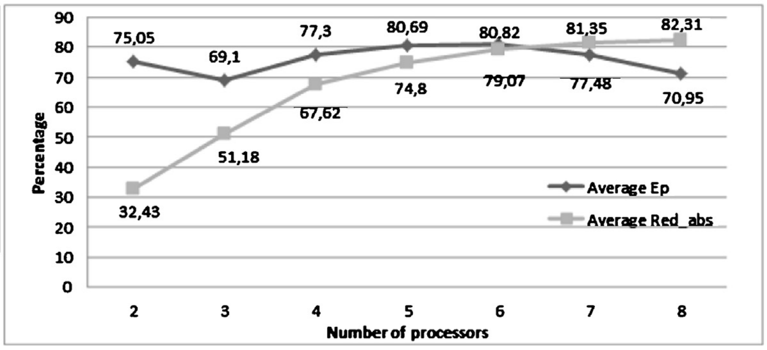
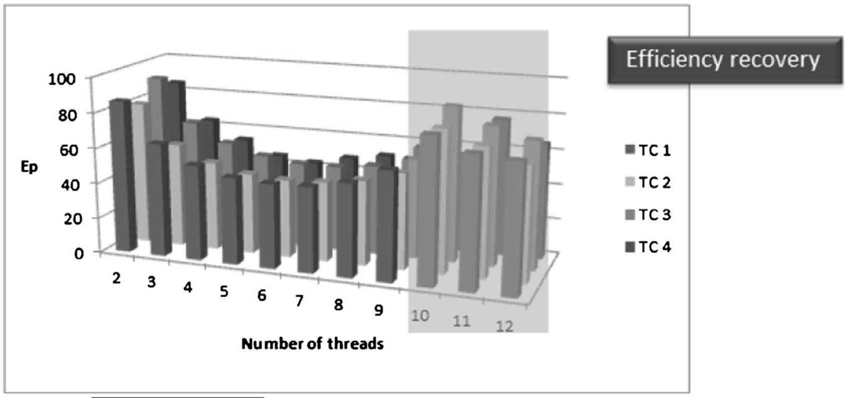


Fig. 10. Efficiency and absolute time reduction on a cluster (System I).

outer loop (see Fig. 3) was the same for every problem regardless of the systems, the values being: 4 iterations for TC1, 7, for TC2, 19, for TC3 and 9, for TC4.

Figs. 6 and 7 show the evolution of the speed-up curves on both platforms. For the cluster (System I) in Fig. 6, the speed-up of the parallel algorithm increases with problem size, especially for those cases where the sequential version demanded a high amount of line-searches. For the multicore platform in Fig. 7, there is a different optimum number of threads depending on the case, with the peaks for 10 or 11 threads. In general terms, after 10 threads there is a plateau because the maximum benefit has been achieved.



	Averages	
N° threads	Ep	Red_abs
2	85,73	41,55
3	62,12	47,07
4	53,3	53,01
5	48,47	58,23
6	46,01	64,12
7	47,03	69,43
8	49,74	74,97
9	57,55	80,57
10	75,92	86,41
11	75,3	87,91
12	68,3	87,77

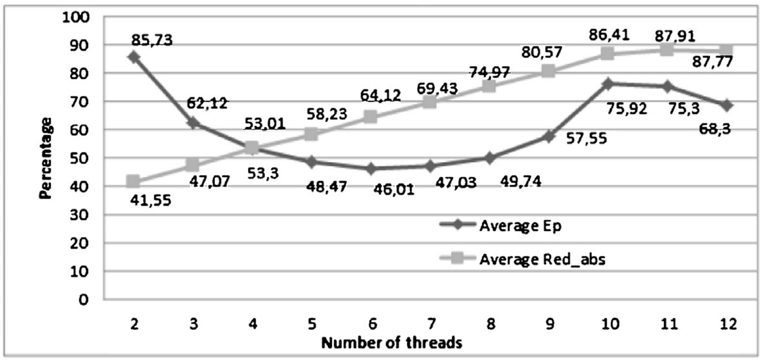


Fig. 11. Efficiency and absolute time reduction on a multicore platform (System II).

3.2.2. Time reduction and efficiency

The behaviour of the speed-up factors S_p for a parallel system of size p is illustrated in Figs. 8 and 9 running under Systems I and II, respectively. The symbols show S_p values calculated for the case studies listed in Section 3.1. For the sake of comparison, the ranges of speed-up factors between the worst and best cases are represented with bars. In Fig. 8 it can be observed that the speed-up factors are closer to the corresponding best case value for any cluster configuration. In contrast, for multi-threading (Fig. 9) the results lie slightly closer to the worst case limit for 2–8 threads, whereas the S_p values approach the best case for configurations with 9–12 threads. In both systems the worst scenario seems to be rather unlikely. This is only a rough estimate, but these illustrations indicate that high speed-up values are attainable with a suitable configuration.

The percentage of absolute reduction of computing time is measured by Eq. (22).

$$Red_{abs} = 100\left(1 - \frac{1}{S_p}\right) \tag{22}$$

Figs. 10 and 11 reveal a significant time reduction in both systems. For a multicore platform, Fig. 11 shows that time reductions reach about 87%, while there is a notorious efficiency increase when 10 threads are employed. This increment is due to the effect of the parallel line-search: the number of calls to the line-search procedure was reduced to a single call per iteration. For a cluster, Fig. 10 shows that although the efficiency is decreasing as the number of processors goes up, the time reduction always keeps an increasing trend.

Apart from the time reduction, we are interested in efficiency. When compared with System I, System II revealed really meaningful efficiency improvements. Fig. 11 shows an efficiency decrease followed by a significant recovery for 10 threads, exhibiting similar behaviour for all cases. The existence of a valley should be taken into account prior to the choice of the number of threads to be adopted.

In Fig. 11 (see the bar plot above) the U-shaped behaviour appears as a result of an efficiency recovery since the number of calls to the parallel line-search procedure has been reduced to its minimum, thus increasing S_p significantly. The shaded rectangle in Fig. 11 highlights this efficiency recovery.

4. Conclusions

There is still a great demand for the development of agile techniques that attractively solve complex problems in short computational times. Nowadays, parallel computing provides a powerful alternative, which is complementary to other approaches, like model reformulation.

In this paper we have described a new methodology called pALSPG, which is a parallel approach of the Spectral Projected Gradient method combined with an Augmented Lagrangian formulation. The parts of the classical sequential ALSPG algorithm that were worth parallelizing were identified, giving way to a new technique. The simultaneous evaluation of the objective function allowed us to determine a satisfactory point in a single call to the line-search procedure.

Better performance has been achieved by means of this approach. Hence, pALSPG becomes more competitive, while maintaining the robustness of the sequential method. For PSE optimization, pALSPG proved to be effective and competent in some problems related to distillation columns. The results presented here clearly demonstrate the feasibility of employing this approach in order to solve problems about optimal sequences of distillation columns. pALSPG allows the efficient use of existing resources not only to boost the speed of computation, but also to accommodate larger problems in a distributed memory environment.

The numerical experiments were here performed by increasing the size of the systems, reaching up to 8 processors for the cluster and up to 12 threads for the multicore computer. Both systems have revealed satisfactory efficiency values. Eventually, if more threads were added, speed-ups might rise. Moreover, the systems exhibited a vital reduction in the computing time, thus making the parallel algorithm really more competitive.

The projection to bigger problems is promising when significant savings are achieved for small examples. It is important to point out that the general growth towards equipment maturity provides a golden opportunity for revamping many existing algorithms in order to take advantage of the availability of an ever increasing computational power.

Notation

ALSPG	Augmented Lagrangian with Spectral Projected Gradient
$C(x)$	set of constraints after the introduction of slack variables
$c_i(x)$	equality constraints
$c_j(x)$	inequality constraints
d_k	search direction in the kth iteration
E_p	percentage of algorithmic efficiency
$f(x)$	nonlinear objective function on any NLP problem
g_k	gradient evaluated in x_k
k	iteration counter
$\mathcal{L}(x, \Lambda, \rho)$	Augmented Lagrangian function
$\mathcal{L}_{\max}$	maximum function value in the last M iterates
l_i	lower bound on the component i of variable x
M	integer that controls the amount of monotonicity
n_i	number of equality constraints
n_j	number of inequality constraints
pALSPG	parallel Augmented Lagrangian with Spectral Projected Gradient
$Proj(x)$	projections onto the feasible region
Red_{abs}	percentage of absolute reduction of computing time
s_k	step from one iterate towards the next one $(x_{k+1} - x_k)$
S_p	speed-up
SPG	Spectral Projected Gradient
t	factor of reduction of the step length
T_f	exclusive computing time of the objective function
T_p	time taken by a parallel system of size p to solve a problem.
T_s	serial time required to solve a problem
T_{Cpt}	computing time
T_{Com}	communication time
u_i	upper bound on the component i of variable x
x_i	continuous ith variable subjected to optimization
x_k	current value of the optimization variable
y_{k-1}	difference between gradients $(y_{k-1} = g_k - g_{k-1})$
z_j	slack variables
λ_k	step length in the kth iteration
Λ	estimate of the vector of Lagrange multipliers
γ	sufficient decrease parameter in the line search procedure
ρ	penalty parameter in the Augmented Lagrangean function

References

Abdel-Jabbar N, Kravaris B, Carnahan C. Partially decentralized state observer and its parallel computer implementation. Ind Eng Chem Res 1998;37(7):2741–60.
Aggarwal A, Floudas C. Synthesis of general separation sequences – non-sharp separations. Comput Chem Eng 1990;14(6):631–53.
Alba E. Parallel metaheuristics: a new class of algorithms. John Wiley and Sons; 2005.
Ardenghi JI, Gibelli TI, Maciel MC. The spectral gradient method for unconstrained optimal control problems. IMA J Numer Anal 2008;29(2):315–31.
Ardenghi JI, Vazquez GE, Brignole NB. Un Software Paralelo para Problemas de Optimización de Gran Tamaño. Mec Comp 2007;26:441–54.

Barzilai J, Borwein JM. Two point step size gradient methods. IMA J Numer Anal 1988;8:141–8.
Bertsekas DP. On the Goldstein–Levitin–Polyak gradient projection method. IEEE Trans Autom Control 1976;21:174–84.
Bertsekas DP, Tsitsiklis JN. Parallel and distributed computations: numerical methods. Prentice Hall; 1997.
Birgin EG, Chambouleyron I, Martinez JM. Estimation of the optical constants and thickness of thin films using unconstrained optimization. J Comp Phys 1999;151:862–80.
Birgin EG, Martinez JM, Raydan M. Spectral projected gradient methods. In: Floudas CA, Pardalos PM, editors. Encyclopedia of optimization. 2nd ed; 2009. p. 3652–9 [Part 19].
Birgin EG, Martinez JM, Raydan M. Non-monotone spectral projected gradient methods on convex sets. SIAM J Optim 2000;10:1196–211.
Birgin EG, Martinez JM, Raydan M. Algorithm 813: SPG-software for convex-constrained optimization. ACM Trans Math Softw (TOMS) 2001;27(3):340–9.
Buzzi-Ferraris G, Manenti F. A combination of parallel computing and object-oriented programming to improve optimizer robustness and efficiency. Computer-Aided Chem Eng 2010;28:337–42.
Cheimarios N, Kokkoris G, Boudouvis AG. An efficient parallel iteration method for multiscale analysis of chemical vapor deposition processes. Appl Numer Math 2013;67:78–88.
Chen Z, Chen X, Yao Z, Shao Z. GPU-based parallel calculation method for molecular weight distribution of batch free radical polymerization. Computer-Aided Chem Eng 2011;29:1583–7.
Cores D, Fung G, Michelena RA. Fast and global two point low storage optimization technique for tracing rays in 2D and 3D isotropic media. J Appl Geophys 2000;45:273–87.
Crema A, Loreto M, Raydan M. Spectral projected subgradient with a momentum term for the Lagrangean dual approach. Comput Oper Res 2007;34:3174–86.
Dennis JE, Schnabel RB. Methods for unconstrained optimization and nonlinear equations. 1st ed. New Jersey: Prentice-Hall; 1983.
Diniz-Ehrhardt MA, Gomes-Ruggiero MA, Martinez JM, Santos SA. Augmented Lagrangian algorithms based on the spectral projected gradient method for solving non-linear programming problems. J Optim Theory Appl 2004;123:497–517.
Domancich AO, Brignole NB, Hoch PM. Structural analysis of reactive distillation columns. Virtual Pro 2009;84:18–38.
Domancich AO, Ardenghi JI, Vazquez GE, Brignole NB. Desempeño de un Algoritmo Espectral para Optimización Rigurosa de Plantas de Procesos Industriales. Mec Comp 2004;23:3047–57.
Egea JA, Vries D, Alonso AA, Banga JR. Global optimization for integrated design and control of computationally expensive process models. Ind Eng Chem Res 2007;46:9148–57.
El-Rewini H, Lewis T. Distributed and parallel computing. Manning Publ; 1997.
Fletcher R. On the Barzilai–Borwein method. Optimization and control with applications. Appl Optim 2005;96:235–56.
Floudas C, Pardalos P. A collection of test problems for constrained global optimization algorithms, Lect Notes in Comp Sci. 455. Springer; 1990.
Gomes-Ruggiero MA, Martínez JM, Santos SA. Spectral Projected Gradient method with inexact restoration for minimization with non-convex constraints. SIAM J Sci Comput 2009;31(3):1628–52.
Grippo L, Lampariello F, Lucidi S. A non-monotone line search technique for Newton's method. SIAM J Numer Anal 1986;23:707–16.
Gropp W, Lusk E, Skjellum A. Using MPI portable parallel programming with the message-passing interface. Cambridge, MA: The MIT Press; 1994.
Hestenes MR. Multiplier and gradient methods. J Optim Theory Appl 1969;4:303–20.
Júdice JJ, Raydan M, Rosa SS, Santos SA. On the solution of the symmetric eigenvalue complementarity problem by the spectral projected gradient algorithm. Numer Algor 2008;47:391–407.
Kang J, Siirola JD, Laird CD. Parallel solution of nonlinear contingency-constrained network problems. Proceedings of the 8th international conference on foundations of computer-aided process design. Computer-Aided Chem Eng, Elsevier 2014;34:705–10.
Kheawhom S. Efficient constraint handling scheme for differential evolutionary algorithms in solving chemical engineering optimization problems. J Ind Eng Chem 2010;16:620–8.
Kraemer K, Kossack S, Marquardt W. Efficient optimization-based design of distillation processes for homogeneous azeotropic mixtures. Ind Eng Chem Res 2009;48:6749–64.
Laird CD, Wong AV, Akesson J. Parallel solution of large-scale dynamic optimization problems. Computer-Aided Chem Eng 2011;29:813–7.
La-Cruz W, Raydan M. Non-monotone spectral methods for large-scale non-linear systems. Optim Methods Softw 2003;18:583–99.
Luengo F, Glunt W, Hayden TL, Raydan M. Preconditioned spectral gradient method. Numer Algorithms 2002;30:241–58.
Molina B, Raydan M. Preconditioned Barzilai–Borwein method for the numerical solution of partial differential equations. Numer Algorithms 1996;13:45–60.
Nocedal J, Wright S. Numerical optimization. Springer-Verlag; 1999.
Piccione PM. Industrial reflections on modelling of fine chemicals and seeds process/product design. Proceedings of the 8th international conference on foundations of computer-aided process design. Computer-Aided Chem Eng, Elsevier 2014;34:212–24.
Powell MJD. A method for non-linear constraints in minimization problems. In: Fletcher R, editor. Optimization. New York: Academic Press; 1969. p. 283–98.
Raydan M. The Barzilai and Borwein Gradient method for the large-scale unconstrained minimization problem. SIAM J Optim 1997;7:26–33.
Wang K, Shao Z, Zhang Z, Chen Z, Fang X, Zhou Z, et al. Convergence depth control for process system optimization. Ind Eng Chem Res 2007;46:7729–38.
Wells C, Glunt W, Hayden TL. Searching conformational space with the spectral distance geometry algorithm. J Mol Struct (Theochem) 1994;308:263–71.
You F, Pinto JM, Capón E, Grossmann IE, Arora N, Megan L. Optimal distribution-inventory planning of industrial gases. I. Fast computational strategies for large-scale problems. Ind Eng Chem Res 2011;50:2910–27.