Parallel Computing for Rolling Mill Process with a Numerical Treatment of the LQR Problem

Computación en paralelo para el proceso de laminación con un tratamiento numérico del problema de LQR

DOI: https://doi.org/10.17981/cesta.01.01.2020.02

Artículo de investigación científica. Fecha de recepción: 11/10/2020 Fecha de aceptación: 23/10/2020

J. A. Gómez Múnera
Corporacion Universidad de la Costa-CUC. Barranquilla (Colombia)
gomezmunera@gmail.com

A. Giraldo-Quintero
Corporacion Estudiantes Universitarios y Profesionales de Marinilla (CORUM). Marinilla (Colombia)
alejo@mail.com

Para citar este artículo:
J. A. Gómez & A. Giraldo-Quintero, "Parallel Computing for Rolling Mill Process with a Numerical Treatment of the LQR Problem", *J. Comput. Electron. Sci.: Theory Appl.*, vol. 1, no. 1, pp. 11–30, 2020. https://doi.org/10.17981/cesta.01.01.2020.02

Abstract— The considerable increase in computation of the optimal control problems has in many cases overflowed the computing capacity available to handle complex systems in real time. For this reason, alternatives such as parallel computing are studied in this article, where the problem is worked out by distributing the tasks among several processors in order to accelerate the computation and to analyze and investigate the reduction of the total time of calculation the incremental gradually the processors used in it. We explore the use of these methods with a case study represented in a rolling mill process, and in turn making use of the strategy of updating the Phase Finals values for the construction of the final penalty matrix for the solution of the differential Riccati Equation. In addition, the order of the problem studied is increasing gradually for compare the improvements achieved in the models with major dimension. Parallel computing alternatives are also studied through multiple processing elements within a single machine or in a cluster via OpenMP, which is an Application Programming Interface (API) that allows the creation of shared memory programs.

Keywords— Automatic control; chemical processes; computer programming; computer techniques; multithreading; parallel algorithms; parallel processing

Resumen— El considerable aumento en el cómputo de los problemas de control óptimo ha desbordado en muchos casos la capacidad de computación disponible para manejar sistemas complejos en tiempo real. Por esta razón, en este artículo se estudian alternativas como la computación paralela, donde el problema se resuelve distribuyendo las tareas entre varios procesadores para acelerar el cómputo y para analizar e investigar la reducción del tiempo total de cálculo incrementando gradualmente los procesadores utilizados en él. Exploramos el uso de estos métodos con un estudio de caso representado en un proceso de laminación, y a su vez haciendo uso de la estrategia de actualización de los valores de las fases finales para la construcción de la matriz de penalización final para la solución de la ecuación de Riccati diferencial. Además, el orden del problema estudiado va aumentando gradualmente para comparar las mejoras logradas en los modelos de mayor dimensión. También se estudian alternativas de computación paralela a través de múltiples elementos de procesamiento dentro de una sola máquina o en un clúster mediante OpenMP, que es una Interfaz de Programación de Aplicaciones (API) que permite la creación de programas de memoria compartida.

Palabras clave— Control automático; procesos químicos; programación informática; técnicas informáticas; multihilo; algoritmos paralelos; procesamiento paralelo

I. Introduction

Parallel computing involves multiple processors (cores) or computers that work as a group to perform tasks. Each core can work on one part of the problem and share information with other processors at varying intervals. In general, it is applied to solve problems that involve a lot of time to get your solution, some related to CFD (Computational Fluid Dynamics), weather prediction, solid materials modeling, dynamics and structural mechanisms, metal casting, among others. In [1] are study FEM (Finite Element Method) applications oriented to CFD problems running in a Beowulf cluster. Parallel architectures have been classified in microcomputers, minicomputers, mainframe and supercomputer [2]. Therefore [3] exposed the Flynn's classification, or others based on memory arrangement and communication between PE (Processing Elements), or in PE connections and memory modules, or in natural PE features, and finally, some specific types of parallel architecture. Those based on memory arrangement can be classified as: (*i*) processors with shared memory, or (*ii*) processors with distributed or local memory.

In shared memory processors in which there are several individual processors and a common memory, each one of them has access to any variable stored in the common memory, i. e. all processors use the same address space: this is the case of computers of coarse granulation like Cray YMP, Cray C90, among others. In distributed processors or local memory, however, each processor has its own private memory, and in this case the address space is not common: all addresses are local and refer to the processor space itself. This case is characteristic of fine-grained memories such as hypercubes, 2D and 3D meshes, trees and clusters [4].

The way to program in parallel computers depends highly on the manner the processors access the memory. In shared memory is widely used the application programming interface OpenMP [5]. For the implementation of parallel programming strategies, it is analyzed in the context of the LQR problem to be used in control problems. The LQR is probably the most studied and used problem in the literature of optimal control. On the other hand, the Hamiltonian formalism has been central in the development of the modern theory of optimal control [6], [7], [8], [9], [10]. The problem of the quadratic linear regulator of finite horizon $n-$dimensional with unmounted controls leads to $2n$ ordinary differential equations. There are well known traditional methods to solve this problem of border conditions, sometimes transforming it into a system of initial conditions by introducing certain additional mathematical objects [11], [12]. For the linear quadratic regulator with infinite-horizon or for bilinear-quadratic regulator there are also some tries to find the missing initial condition for the costate variable from the data of each particular problem, whereas in turn allows to integrate the Hamiltonian equations on-line with the related control system [13].

A the restricted-control context may lead to irregular intervals (non-regular optimal control problems), where the solution to the control problem is not analytically available and not standard recipes [7], [14], [15], [16], [12], [17]. Since the early sixties, the Pontryagin Maximum Principle (PMP) has been the conventional theoretical setting to treat such non regular situations. To solve the constrained LQR one takes advantage of certain mathematical relationships that exist between the PMP and the Hamiltonian formalism. The underlying theoretical results [18], [19], [20], are the basis of this article. The treatment can be paraphrased as follows: in general, the optimal solution of a 'simple' LQR problem with bounded controls is the saturation of the solution of another unrestricted LQR, this last problem with the same dynamics and the same cost function as the original, but with different initial conditions and subject to a final quadratic penalty with final penalty matrix different from the original matrix. In this context, a 'simple' LQR problem is called when has a single regular period.

In [21], strategies were developed "off-line" and "on-line" (based on the gradient method) to detect the 'news' initial conditions $\tilde{x_0}$ and the final penalty matrix $\tilde{S}$. Algebraic formulas were given here to calculate the partial derivatives of cost with respect to $\tilde{S}$ and $\tilde{x_0}$, with the main aim of avoiding the numerical integration of states trajectory, control and/or cost, and lower computational effort.

The main contribution of this article is the use of parallel programming strategies to reduce the total computation time and analyze the improvements obtained by gradually increasing the order of the model studied in the rolling mill process. It is achieved that the time of calculation of the algebraic formulas for calculate the partial derivatives with respect to the new parameters are obtained in smaller times allowing the total reduction. The search for these parameters instead of the originals used in [21], implied in themselves a significant reduction in the dimension of the unknowns and therefore a decrease of the computational effort, when adding the strategies of parallel computing for shared memory allowed to make a treatment of the system increasing its dimension.

This scheme update similarly the (ρ, μ), and also the times of "switching" τ_i where the control is bounded, while the total cost is reduced by the gradient method [22], [23], [5]. In general, it is expected that the solution producing the method will be suboptimal, although in some cases it is possible to achieve the optimal solution numerically.

This work is then as a complement to the scheme for obtaining control [19], [20], and in some ways can be thought of as an alternative to sophisticated programming approaches [24], which depend on the discretization adopted in time and space.

II. Regular Lqr Optimal Control Problem

The classical solution to the optimal control problem for linear systems, the finite-horizon, time-constant formulation of the LQR problem with free final states of dimension n and unconstrained controls attempts to minimize the (quadratic) cost as (1):

$$\mathcal{J}(u) = \int_0^{t_f} [x'(\tau)Qx(\tau) + u'(\tau)Ru(\tau)]d\tau + x'(t_f)Sx(t_f) \quad (1)$$

With respect to all the admissible (here piecewise-continuous) control trajectories u: [0, t_f] → $\mathbb{R}^m$ of duration t_f. The control strategies affect the $\mathbb{R}^n$-valued states x through some initialized, autonomous, linear dynamical constraint of the type (2):

$$\dot{x} = Ax + Bu := f(x,u); x(0) = x_0, \quad (2)$$

Where the "control" $u(t)$ (It is understood as a trajectory of vectors $\mathbb{R}^m$) which influences the variables to be controlled or "states" of the system (represented by x, where $x(t)$ is a vector of the Euclidean space n-dimensional denoting by $\mathbb{R}^n$ for each t).

The matrices in (1), (2) are assumed to have the following properties: $Q \geq 0$ and $S \geq 0$ are positive semi-definite $n \times n$ matrices, $R > 0$ is $m \times m$ and positive definite, the dimensions of A is $n \times n$, with B is $n \times m$, and the pair (A, B) is controllable. The objects concerning the functional are of the type (3), (4):

$$L(t,x,u) = x'Qx + u'Ru \quad (3)$$

$$\mathcal{K}\left(x(t_f)\right) = x'(t_f)Sx(t_f), t_f < \infty \quad (4)$$

Where L is known as the 'Lagrangian' L of the cost and $K(x(t_f))$ represents a final penalty. With these conditions, the Hamiltonian of the problem $H(t, x, \lambda, u)$ is expressed as in (5).

$$\begin{aligned} \mathcal{H}(t,x,\lambda,u) &= L(t,x,u) + \lambda' f(t,x,u) \\ &= x'Qx + u'Ru + \lambda'(Ax + Bu) \end{aligned} \quad (5)$$

And is seen as a mapping function $[t_0, t_f] \times \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathrm{R}$. Assuming the regular Hamiltonian $H(t, x, \lambda, u)$ &, i.e. differentiable with respect to u for each pair (x, λ) in the condition $\partial H/\partial u = 0$, the optimal control problem occurs when u takes the explicit control value according to (6).

$$u^0(t, x^*(t), \lambda^*(t)) = u^*(t) = -\frac{1}{2} R^{-1} B' \lambda^*(t) \qquad (6)$$

Which is usually called 'the H-minimal control'. Finding the optimal control for a regular problem [7], [25], [12] requires to solve the two-point boundary-value problem known as the 'Hamilton Canonical Equations' (HCEs), defined as (7) and (8) (see [9] for general problems and [12], p. 406 for the case with free final state).

$$\dot{x} = \left(\frac{\partial \mathcal{H}^0}{\partial \lambda}\right)' (x, \lambda) \triangleq \mathcal{F}(x, \lambda); x(0) = x_0 \qquad (7)$$

$$\dot{\lambda} = -\left(\frac{\partial \mathcal{H}^0}{\partial x}\right)' (x, \lambda) \triangleq -\mathcal{G}(x, \lambda); \lambda(t_f) = \frac{\partial \mathcal{K}}{\partial x}\left(x(t_f)\right) \qquad (8)$$

Which define a system of ordinary differential equations of dimensions $2n$, where $0(x, \lambda)$. The Equation (9) is called the 'minimized Hamiltonian', stands for:

$$\mathcal{H}^0(x, \lambda) := \mathcal{H}\left(x, \lambda, u^0(x^*(t), \lambda^*(t))\right) \qquad (9)$$

The optimal trajectories of the state and costate are solutions of the Hamiltonian canonical equations in the phase space given by (7), (8). for this case un particularly of the LQR problem, if the optimal Hamiltonian is given by (9), the result would be (10), (11):

$$\dot{x} = Ax + Bu^0(x, \lambda) = Ax - \frac{1}{2} W\lambda \qquad (10)$$

$$\dot{\lambda} = 2[\dot{P}(t)x + P(t)\dot{x}] = -2Qx - A'\lambda \qquad (11)$$

Where $W \triangleq BR^{-1}B'$

The solution in feedback form to the unrestricted regular problem, is based in turn on the solution of $P(\cdot)$ 'Riccati Differential Equation' (RDE) showed in (12):

$$\dot{P} = PW(t)P - PA - A'P - Q \qquad (12)$$

With the final boundary condition $P(t_f) = S$. By combining (6) with the optimal solution (8), the control action in open-loop of (13) (depend of system co-states) becomes an equation with state dependence:

$$u^*(t) = -R^{-1}(t)B'(t)P(t)x^*(t) \tag{13}$$

Equivalently to the equation (13), the optimal feedback law in the regular case is (14):

$$u_f(t,x) = -R^{-1}B'P(t)x \tag{14}$$

When the manipulated variable 'u^0 values are restricted, then the global regularity of the Hamiltonian cannot be guaranteed. An alternative search for the optimal control strategy is necessary.

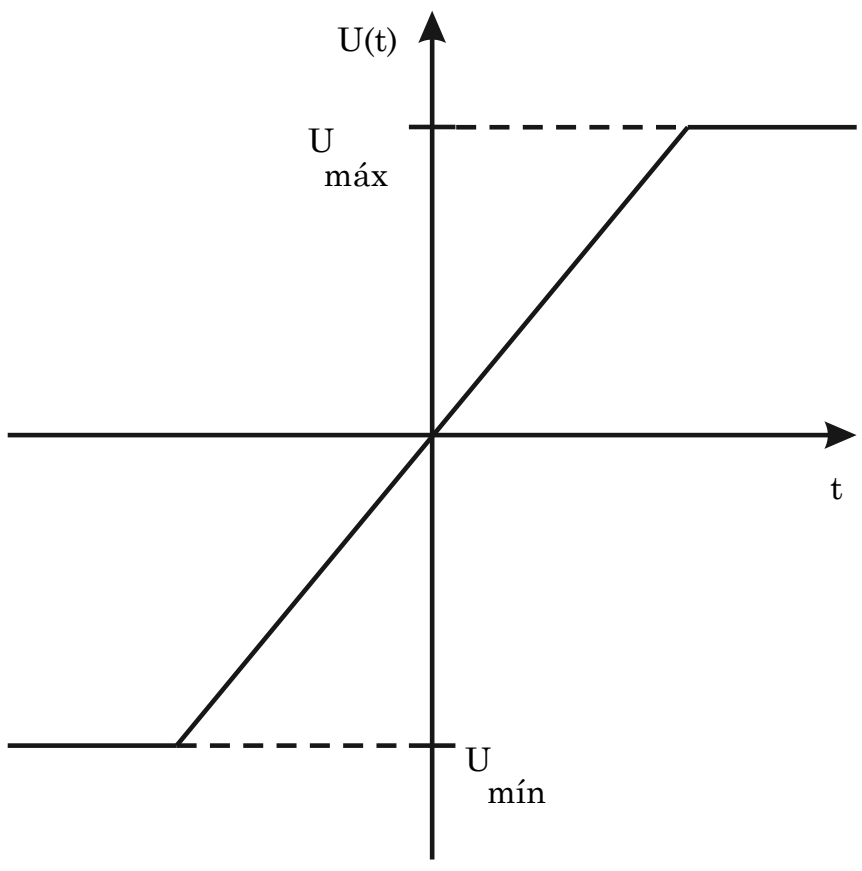

Fig. 1. Bounded Control for dimension $m = 1$.
Source: Author(s).

The Hamiltonian matrix for the original problem with the invariant time is given in (15):

$$\mathbf{H}:=\begin{pmatrix} A & -W/2 \\ -2Q & -A' \end{pmatrix} \tag{15}$$

And the associated fundamental matrix (16):

$$\mathbf{U}(t):= e^{-\mathbf{H}t} \tag{16}$$

The matrix $\mathbf{U}(t)$ is the dimension $2n \times 2n$. It will be partitioned into 4 sub-matrices as shown by (17):

$$\begin{pmatrix} \mathbf{U}_1(t) & \mathbf{U}_2(t) \\ \mathbf{U}_3(t) & \mathbf{U}_4(t) \end{pmatrix} := \mathbf{U}(t) \tag{17}$$

In previous results of LQR problem in the Hamiltonian form [25], [8], [26], [12] other mathematical objects appear which will be useful in understanding of the restricted solution, i.e. the matrices $\alpha(t_f, S)$ and $\beta(t_f, S)$ (18), (19). These two auxiliary matrices are defined as:

$$\begin{pmatrix} \alpha(t_f - t, S) \\ \beta(t_f - t, S) \end{pmatrix} = \mathbf{U}(t_f - t) \begin{pmatrix} I \\ 2S \end{pmatrix} \tag{18}$$

$$\begin{aligned} \begin{pmatrix} \alpha(t_f, S) \\ \beta(t_f, S) \end{pmatrix} &\triangleq \mathbf{U}(t_f) \begin{pmatrix} I \\ 2S \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{U}_1(t_f) + 2\mathbf{U}_2(t_f)S \\ \mathbf{U}_3(t_f) + 2\mathbf{U}_4(t_f)S \end{pmatrix} \end{aligned} \tag{19}$$

The matrices allow us to calculate, the solution $P(\cdot, t_f, S)$ 'RDE0 through the expression (20) [8].

$$P(t, t_f, S) = \frac{1}{2}\beta(t_f - t, S)\left[\alpha(t_f - t, S)\right]^{-1} \forall t \in [0, t_f] \tag{20}$$

The matrices α, β are associated to the boundary conditions of the HCEs by the relations expressed (21).

$$\begin{aligned} x(t_f) &= \alpha(t_f, S)^{-1} x(0), \\ \lambda(0) &= \beta(t_f, S) x(t_f), \\ \lambda(t_f) &= 2S x(t_f). \end{aligned} \tag{21}$$

III. Bounded control and the pontriagyn principle

In the most of practical applications the manipulated variable in can only assume a bounded set of values [27], [28], [29]. The expression 'manipulated' indicates that an instrument or a person assigns a value to a signal generated by physical means also named 'input', and therefore this value can only take that physically realizable, in some cases how in the amount of fuel supplied to a motor, temperature, current, voltage, among others, cannot take arbitrary values. The manipulated variable 'u^0 can move inside and on the boundary of some bounded subset of a metric space M ⊆ R (22). Fig. 1 shows the saturation of the manipulated variable for $m = 1$.

$$u(t) \in \mathbb{U} := [u_{min}, u_{max}] \tag{22}$$

The qualitative features of optimal control solutions to bounded problems are significantly different from those of unbounded ones [7], [18], [30], [9].

Problems with bounded in the control variable are treated in this article with the intention of exploiting theoretical results exposed in [18], [19], [20]:

In [19] a strategy is used to work with linear problems when you have constraints on the manipulated variable through the so-called phantom problem based on the approximation of unknown parameters, in [31] the problem is extended to work with non-linear systems.

The generation of sub-optimal control strategies based on the approximation of unknown parameters $\{\hat{S}, \hat{x}_0\}$ has been recently published in [18]. In [21] algebraic formulas are used to calculate the partial derivatives of the total cost with respect to the coefficients of the matrix $\hat{S}$ and initial conditions $\hat{x}_0$, without the need to generate state trajectories or evaluate the cost. In this case, the number of variables to be updated was $((n + 3)\, n)/2$, which for applications of great dimension can be impractical. In [19], [20] it was possible to reduce the number of unknowns by focusing on updating the final phase values ρ and μ for the construction of the final penalty matrix S, passing in this case to update $2n + 2$ variables, still avoiding the ODEs numerical integration, as in the presented results of [21]. A direct consequence is the reduction of computational effort required to minimize the total cost. This article seeks to take advantage of the reduction of the computational effort obtained in [19], [20], and also reduce the total computation times for the calculation of the numerical strategy achieving reach at least sub-optimal results, all through parallel processing with the parallel programming model for shared memory OpenMP [32].

A. *Control Seed through the Unbounded Final Phase Values and Auxiliary Objects*

The following type of feedback control laws (23) for a regular interval, will be frequently used in the sequel to obtain the sub-optimal control of the problem

$$\tilde{u}(t) := \begin{cases} u_{min}, & \forall t \in [0, \tau_1) \\ -R^{-1}B'P(t, \rho, \mu)x(t), & \forall t \in [\tau_1, \tau_2) \\ u_{max}, & \forall t \in [\tau_2, t_f] \end{cases} \tag{23}$$

Where $\tilde{u}(t)$ is an abbreviated notation for $\tilde{u}_{\rho,\mu,\tau 1,\tau 2}(t)$, which will be used to indicate that the feedback law is associated to the parameters (ρ, μ, $\tau 1$, $\tau 2$). The strategy proposes variations of the final states (final conditions) ρ and μ in such a way that the matrix $\tilde{S}_i \rightarrow \tilde{S}$. The approach should take into account the following considerations:

- The initial matrix $\hat{S}_0$ of $n \times n$ will be constructed from the optimal state and the costate trajectories of the unconstrained problem (1), (2) by applying the feedback (14). The costate verifies at that and then by denoting $\rho_{seed} = x(tf)$, $\mu_{seed} = \lambda(tf)$, the seed matrix $\hat{S}_0$ is defined as (24):

$$\tilde{S}_0 := \frac{1}{2}\frac{\mu_{\text{seed}}\mu'_{\text{seed}}}{\rho'_{\text{seed}}\mu_{\text{seed}}} \tag{24}$$

- A 'seed' strategy is taken to start the iterative method below as in (25), namely:

$$u_{\text{seed}}(t) := \begin{cases} u_{min} \text{ if } -R^{-1}B'P\left(t,\tilde{S}_0\right)x(t) & \leq u_{min} \\ u_{max} \text{ if } -R^{-1}B'P\left(t,\tilde{S}_0\right)x(t) & \geq u_{max} \\ -R^{-1}B'P\left(t,\tilde{S}_0\right)x(t) & \text{otherwise} \end{cases} \quad (25)$$

The state trajectory corresponding to the control u_{seed} and starting at x_0, *i.e.* x_{useed}, will be denoted as x_{seed}. From the seed control and state trajectories simulated for the new final matrix $\hat{S}_0$, the first values are obtained for the saturation times $\tau_{1,0} \leq \tau_{2,0}$, can be detected if they exist.

- $P(t, \rho, \mu)$ is the solution to the RDE (12), with final condition $P(tf) = \hat{S}$. The following identity will also be used ($\eta := \rho$ or μ) from (20) is obtained (26):

$$\begin{aligned} \frac{\partial P(t,\eta)}{\partial \eta} &= \frac{\partial\left[\frac{1}{2}\beta\left(t_f - t,\eta\right)\left[\alpha\left(t_f - t,\eta\right)\right]^{-1}\right]}{\partial \eta} \\ &= \frac{1}{2}\left[\beta_\eta \alpha^{-1} - \beta\alpha^{-1}\alpha_\eta\alpha^{-1}\right]\left(t_f - t,\eta\right) \\ &= \frac{1}{2}\left[\beta_\eta - 2P(t,\eta)\alpha_\eta\right]\alpha^{-1} \end{aligned} \quad (26)$$

Where α, $\alpha_\eta := \partial\alpha/(\partial\eta)$, $\beta\eta := \partial\beta/(\partial\eta)$ should be evaluated at ($tf - t$, ρ, μ), and from (17), (19), the partial derivatives $\alpha\eta$, $\beta\eta$ result in (27).

$$\alpha_\eta = 2\mathbf{U}_2\frac{\partial S}{\partial \eta}, \beta_\eta = 2\mathbf{U}_4\frac{\partial S}{\partial \eta} \quad (27)$$

1) *The partial derivatives of the cost*

Previous results [18] indicate that the optimal control $u^*(\cdot)$ for the original problem can be obtained by saturating the optimal control $\hat{u}(\cdot)$ corresponding to the unrestricted ˆproblem (phantom system). But actually it is not necessary to determine the whole $\hat{u}(\cdot)$ trajectory since we know it saturates outside $I = [\tau_1, \tau_2]$, and also we know the values that $\hat{u}(\cdot)$ assumes for $t \notin [\tau_1, \tau_2]$. So it is enough to calculate τ_1, τ_2 and $\hat{P}(t)$, for $t \in I$ to define the whole $u*(\cdot)$ strategy.

It is known that the total cost $J(\tilde{u})$ is differentiable as a function of the variables (ρ, μ, τ_1, τ_2) [33]. It is time-partitioned here for convenience into 4 parts according to (28).

$$J(\rho,\mu,\tau_1,\tau_2) := \mathcal{J}(\tilde{u}) = J_1 + J_2 + J_3 + J_4 \quad (28)$$

Where J_1 (29) accounts for the trajectory cost associated with the saturation period [0, τ1], J_2 (30) with the regular interval [τ_1, τ_2], J_3 (31) with [τ_2, t_f], and J_4 (32) measures the final penalty. Consequently, the partial costs are:

$$J_1 = Ru_{min}^2\tau_1 + \int_0^{\tau_1} x'(t)Qx(t)dt \tag{29}$$

$$J_2 = x'(\tau_1)P(\tau_1)x(\tau_1) - x'(\tau_2)P(\tau_2)x(\tau_2) \tag{30}$$

$$J_3 = Ru_{max}^2\left(t_f - \tau_2\right) + \int_{\tau_2}^{t_f} x'(t)Qx(t)dt \tag{31}$$

$$J_4 = x'\left(t_f\right)Sx\left(t_f\right) \tag{32}$$

The derivatives D_η with respect to the variables ρ and μ (denoted by the variable η), and switching times τ_1 y τ_2 are obtained as [19], [20], and are expressed as (33).

$$D_\eta J_1,\ D_\eta J_2,\ D_\eta J_3,\ D_\eta J_4 \tag{33}$$

The Equation (34) for the switching time τ_1:

$$D_{\tau 1} J_1,\ D_{\tau 1} J_2,\ D_{\tau 1} J_3,\ D_{\tau 1} J_4 \tag{34}$$

And for the switching time $\tau 2$ (35) so that:

$$D_{\tau 2} J_1,\ D_{\tau 2} J_2,\ D_{\tau 2} J_3,\ D_{\tau 2} J_4 \tag{35}$$

The partial derivatives of costs with respect to the Final Phase values ρ and μ referenced (33) have intrinsically involved the variations of the RDE represented (26) [19], [20], The partial derivatives of P can be calculated (26)-(27), giving (36) and (37):

$$\frac{\partial \tilde{S}_{kl}}{\partial \rho_i} = \frac{1}{2}(\mu_k \mu_l)\left(-\frac{\mu_i}{(\rho'\mu)^2}\right) \tag{36}$$

$$\frac{\partial \tilde{S}_{kl}}{\partial \mu_j} = \frac{1}{2}\left[\frac{Z_{j_{(kl)}}}{(\rho'\mu)} - \frac{(\mu_k \mu_l)\rho_j}{(\rho'\mu)^2}\right] \tag{37}$$

Where $Z_{j(kl)}$ is given according (38):

$$Z_{j(kl)} := \frac{\partial \mu_k \mu_l}{\partial \mu_j} = \begin{cases} 2\mu_j & \text{si} \quad k = l = j \\ \mu_k & \text{si} \quad l = j \text{ y } k \neq j \\ \mu_l & \text{si} \quad k = j \text{ y } l \neq j \\ 0 & \text{si} \quad k \neq j \text{ y } l \neq j \end{cases} \tag{38}$$

2) *Updating the parameters*

First approximations $\tau_{1,0}$, $\tau_{2,0}$ to the optimal saturation points τ_1, τ_2 become available after simulating the state trajectory x_{seed} as described (III-A). For $t \in [0, t_f]$ the control is set to (39):

$$\tilde{u}_0(t) \equiv u_{seed}(t) \tag{39}$$

The parameters (ρ, μ, τ_1, τ_2) are then updated to construct successive control strategies $\tilde{u_j}$; with $j = 1, 2, \ldots$ seeking to decrease the value of the total cost as shown (40):

$$\mathcal{J}(\tilde{u}_{j+1}) \leq \mathcal{J}(\tilde{u}_j) \leq \cdots \mathcal{J}(u_{\text{seed}});\ j = 1, 2, \ldots \tag{40}$$

Using the gradient method (41), (42):

$$\begin{cases} \rho_j = \rho_{j-1} - \gamma_j \dfrac{\partial J}{\partial \rho}(\rho_{j-1}, \mu_j, \tau_{1,j}, \tau_{2,j}) \\ \mu_j = \mu_{j-1} - \gamma_j \dfrac{\partial J}{\partial \mu}(\rho_j, \mu_{j-1}, \tau_{1,j}, \tau_{2,j}) \end{cases} \tag{41}$$

$$\begin{cases} \tau_{1,j} = \tau_{1,j-1} - \gamma_j \dfrac{\partial J}{\partial \tau_1}(\rho_j, \mu_j, \tau_{1,j-1}, \tau_{2,j}) \\ \tau_{2,j} = \tau_{2,j-1} - \gamma_j \dfrac{\partial f}{\partial \tau_2}(\rho_j, \mu_j, \tau_{1,j}, \tau_{2,j-1}) \end{cases} \tag{42}$$

Until the final phase values converge or a practical stop decision is made. The value of γ_j is a positive small number real, that measuring the portion of the gradient vector to be applied in each iteration, selected and tuned by the user.

Other sophisticated versions of the gradient method such as the method of Fletcher and Reeves [22], [23], corresponding to the conjugate search directions are calculated sequentially starting from (43):

$$s^0 := -\nabla J(r^0) \tag{43}$$

With r denoting the parameters involved (ρ, μ, τ_1, τ_2), and r^0 contains their seed values. s^0 is the initiating search direction and ∇ is the (row) gradient operator. The updating equation for the search direction (prescribed by this technique) is according (44).

$$s^{j+1} = -\nabla J\left(p^{j+1}\right) + s^j \frac{\nabla J\left(p^{j+1}\right)\nabla' J\left(p^{j+1}\right)}{\nabla J(p^j)\nabla' J(p^j)},\ j = 1, 2, \ldots \tag{44}$$

And the updating of parameters is conducted as follows (45):

$$(\rho, \mu, \tau_1, \tau_2)_{j+1} = (\rho, \mu, \tau_1, \tau_2)_j + \gamma_j s^j \tag{45}$$

IV. Parallel computing through programming shared memory use OpenMP

OpenMP (Open Multi-Processing) is an Application Programming Interface (API) that supports multi-platform shared memory multiprocessing programming, whose feature is based on facilitating parallel programming of shared memory in multi-core machines. OpenMP is not itself a programming language, it corresponds to a library that can be added to languages like C, C ++ or Fortran to describe how the work can be distributed among the threads that are executed in the different processors or cores available.

The API is designed to allow incremental approximations to parallelize existing code, in which parts of a program are paralleled, probably following a logical sequence for it.

To work with OpenMP, you must initially open a parallel region consisting of a block of code executed by multiple processors simultaneously. Parallelization through sections and loops is referred to as a shared workspace construction and these must be within a parallel region respectively in which all processors find the statement together or none of them. In Fig. 2 shows how the works are distributed in the processors when the parallel region is started and the union and synchronization of the same when finished.

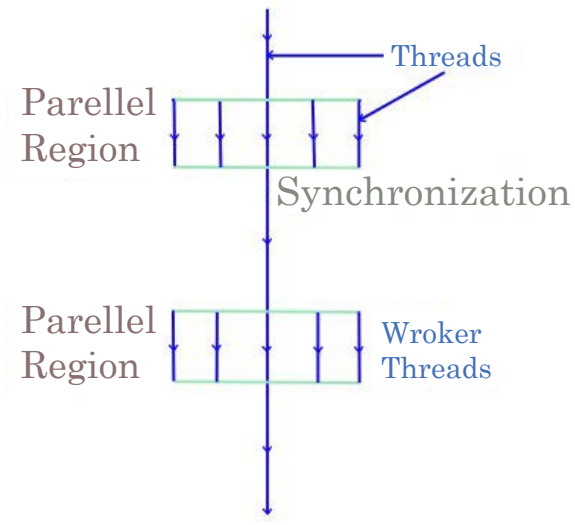

Fig. 2. Fork and Join model.
Source: Authors(s).

A. *Coding Programming by Sections*

This type of parallelism is done by means of individual blocks of code, which are distributed on all the processors available, or according to the sections or areas that are to be parallelized. The functionality is then to do the distribution of independent work units. The syntax for encoding sections in C/C++ is as follows:

```
# pragma omp parallel        [clause(s)]
{
 #pragma omp sections
{
 #pragma omp section
{
        <codeblock 1>
    }
       #pragma omp     section
{
<code block 2>
} /*-- end of section --*/
.
.
.
 }/*-- end of       sections --*/
}/*-- end of        parallel    region --*/
```

B. *Coding Programming by Loops for*

To use this type of parallelization, loop construction causes the next loop iterations to run in parallel. At runtime, the loop iterations are distributed through the processors. The functionality of this is to distribute the processors over the iterations involved in the programming. The syntax for encoding loops 'for' in C/C++ is:

```
# pragma omp parallel        [clause(s)]
{
#pragma omp for
{
        <codeloop   for>
}/*-- end for --*/
}/*-- end of       parallel     region --*/
```

V. Application and numerical results. A typical linearized model situation: the rolling mill.

The case-study models a rolling mill process as described in [21], [34], led to the form of an invariant linear control system [19] (46).

$$\dot{x} = Ax + Bu \tag{46}$$

Where the $n \times n$ matrix A and the column n-vector B take the form according (47) and (48).

$$A = (a_{ij}): \begin{cases} a_{ii} = V_0/h - a_1 a_{i,i+1} = -V_0/h_i \\ \text{with } i = 1, \dots, n-1 \\ a_{n,n-1} = V_0/h; a_{nn} = -(a \pm V_0/h)' \\ \text{all remaining elements equal to 0} \end{cases} \quad (47)$$

$$B = (b_i) = \left[\frac{a}{V_0}(\theta_0 - \theta_a)\exp(-az_i/V_0); i = 1, \dots, n\right] \quad (48)$$

The following values for the parameters determined by (49) were investigated.

$$V_0 = h = 1; a = 1.001 \quad (49)$$

The discretized, ODE version (46) was numerically confirmed to be an acceptable approximation [21]. The initial state $x_0 = x(0)$ used for simulation of the system defined by (46)-(48) was as in (50).

$$x_i(0) = 100; i = 1, \dots, n \quad (50)$$

With the following values for the reference temperatures giving (51) (in °C):

$$\theta_a = 20, \theta_0 = 700 \quad (51)$$

The cost objective function or index of the LQR problem was assigned the following parameters (52).

$$t_f = 0.5, Q = 0.05I_n, R = 100, S = 15I_n, \quad (52)$$

With $n = 10$.

The bounds imposed on control values were chosen here as in (53).

$$[u_{min}, u_{max}] = [-1.8, -0.8] \quad (53)$$

Just to illustrate a case where the costs of the unrestricted problem and the first approximation to the restricted one (the 'seed') are different enough to justify looking for better approximations and to obtain a significant cost reduction through the implementation of the method. Indeed, the seed trajectory was calculated by applying (25), from which the initial switching times were detected: $\tau_{1,0} = 0.0413$, $\tau_{2,0} = 0.0865$, and the total cost resulted $J_{seed} = 1.1316 \times 10^6$, while the unbounded cost was $Junbounded = 4.3338 \times 105$.

Fig. 3 the seed control trajectory (dotted line) and its evolution towards the optimal control (solid line) are illustrated, also the optimal unbonded control (discontinued line). In this case-study, the optimal control corresponds to the maximum value applied over the entire time horizon, mainly due to the high values assigned to R and S in the original formulation of the cost, the optimal control resulted $u*(t) \equiv -0.8 = u_{max}\ \forall\ t$.

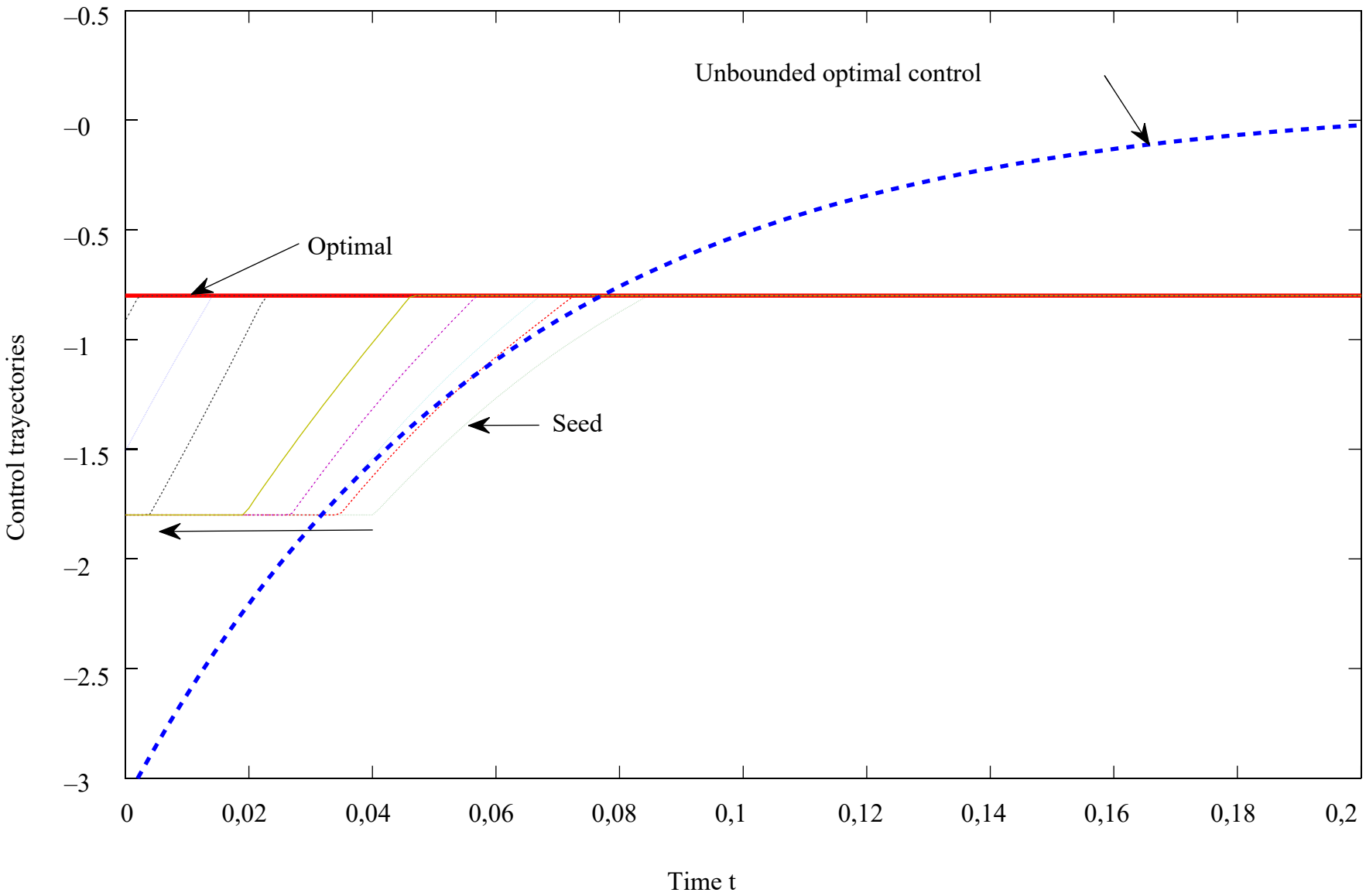

Fig. 3. Progress of control the trajectories with iterations until the final phase value converge and reaching the optimal control.
Source: Author(s).

The optimality of the obtained control trajectory $u*(t) \equiv -0.8 = u_{max}\ \forall\ t$ was checked by calculating:

(*i*) the solution to the Hamiltonian equations (7), (8), backwardly from ($\rho*$, $\mu*$), which reproduced the initial state condition, and (*ii*) the Hamiltonian along the trajectory, which resulted in $H(x*(t), \lambda*(t), u*(t)) \equiv 2.2191 \times 10^6$, constant as expected.

The final values for the swictching times are: $\tau_1 = -0{,}0198$, $\tau_2 = -0{,}0028$, these negative values prove the loss of the regular period.

A. *Parallel Computing for the Rolling Mill process*

To perform the operations involved in the proposed method, the Armadillo library was used [35], is a high-quality library for linear algebra operations in language C++.

The problem was worked on shared memory, and tests were performed both by parallelizing sections and by parallelizing loops.

1) *Program Parallelization Through Sections*

To work on improving the approximation achieved with the rolling mill process (section V) is used different dimensions of the model, where several spatial discretizations were carried out gradually increasing the dimensions of the same. Keeping the value of $L = 10$, he values of the dimensions n and the discretization step h were modified to obtain the different x_i, α_{ij} and b_i.

Parallelization instructions are performed using OpenMP. The fractions of calculation that can be paralleled by working in this way are limited by the 4 derivatives of the corresponding cost (J_1, J_2, J_3, J_4) according to each one of the variables involved, in the variable η synthesized (33), for switching times τ_1 (34) and τ_2 (35) respectively. Therefore, the portion of the program parallelized corresponds to a distribution by sections of each of them as described in the section IV-A, the ones that imply the greatest computational load are those corresponding to the calculate the final phase values. Therefore, is obtained a greater saving of calculation from its parallelization, in this case with a maximum of 3 sections, due to the value 0 in the derivative $D_\eta J_1$. The results working with $p = 2$ locals processors can be seen in the Table I, with t_1 and t_2 given in seconds, where $\Delta t_{12} = t_1 - t_2$.

Fig. 4 shows the speedup and the total gain of time obtained to parallelize with $p = 2$ local processors, although a decrease of speedup is evidenced with the increase in the dimension n of the problem that consequently leads to a decrease in the efficiency of the system when talking about a parallel architecture, the total calculation time is reduced considerably, precisely in view of the increase of the dimension.

TABLE I.
MEASURES BY ITERATION OF THE PARALLEL PROGRAM WITH OMP SECTIONS.

n	S_p	E_f	t_1	t_2	Δt_{12}
20	1.996	99.8%	80.529	40.343	40.186
50	1.979	98.95%	206.08	104.11	101.97
100	1.878	93.89%	482.95	257.17	225.78
150	1.66	83%	1062.9	639.83	423.14
200	1.52	76%	1845.2	1213.3	631.94

Source: Author(s).

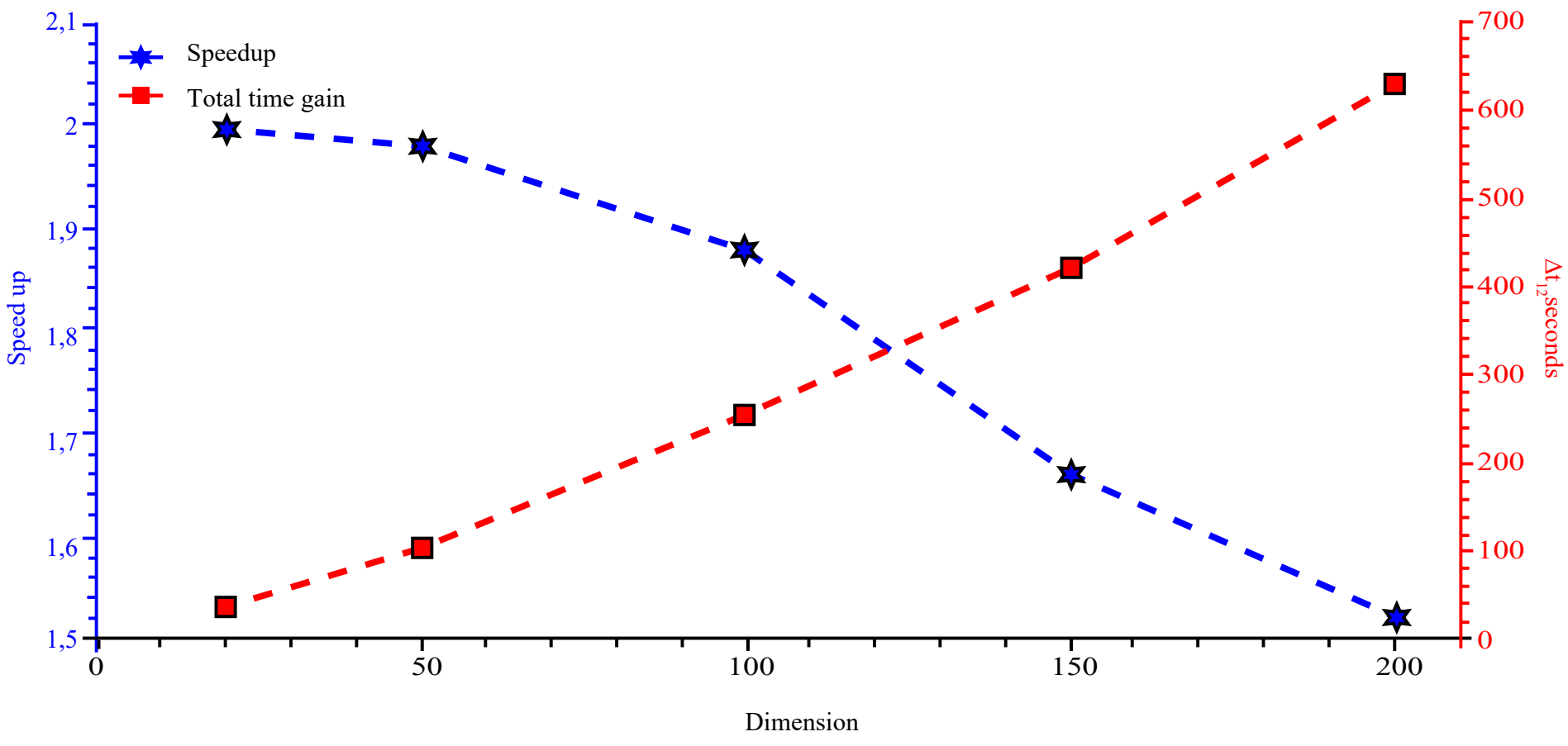

Fig. 4. Speedup and total gain for the final phase update problem with sections and locals processors $p = 2$.
Source: Author(s).

2) *Program Parallelization Through Loops*

Program Parallelization through loops is performed on the partial derivatives of the final penalty matrix with respect to the final phase values involved (27), making for this a distribution of the vector elements that make up the final phase values ρ_i, μ_j (36), (37). Therefore, the portion of the program parallelized corresponds to a distribution on the processors of the iterations of the Final.

3) *Phase variables with the procedure described in section IV-B.*

The results of parallelization working with diverse number of processors in the cluster (currently the cluster available in CIMEC is called 'coyote') and different dimension of the problem of n = 10,20,50 and 100. To run the program in the cluster it is necessary to create a job.

In the tables are presented the results obtained for the different dimensions, in the Table II for n = 10, in the Table III for n = 20, in the Table IV for n = 50 and in the Table V for n = 100.

The Table VI shows the total computation gain when applying parallel programming strategies on the program of updating the values of Final Phase for the different dimensions tested and also with increasing gradually the processors involved in the calculation.

TABLE II.

MEASURES BY ITERATION OF THE PARALLEL PROGRAM WITH OMP FOR IN THE CLUSTER WITH N = 10 AND T_1 = 80.9, THE MEASURES OF TIME IN SECONDS.

p	t_2	t_4	t_6	t_8	S_p	E_f
1	–	–	–	–	–	–
2	41	–	–	–	2	99%
4	–	25	–	–	3	82%
6	–	–	16	–	5	82%
8	–	–	–	17	5	61%

Source: Author(s).

TABLE III.

MEASURES BY ITERATION OF THE PARALLEL PROGRAM WITH OMP FOR IN THE CLUSTER WITH N = 20 AND $T1$ = 165, THE MEASURES OF TIME IN SECONDS.

p	t_2	t_4	t_6	t_8	S_p	E_f
1	–	–	–	–	–	–
2	84.5	–	–	–	2.0	97%
4	–	43.7	–	–	3.8	94%
6	–	–	35.8	–	4.6	77%
8	–	–	–	27.6	6.0	75%

Source: Author(s).

TABLE IV.

MEASURES BY ITERATION OF THE PARALLEL PROGRAM WITH OMP FOR IN THE CLUSTER WITH N = 50 AND $T1$ = 494, THE MEASURES OF TIME IN SECONDS.

p	t_2	t_4	t_6	t_8	S_p	E_f
1	–	–	–	–	–	–
2	294	–	–	–	1.7	84%
4	–	198	–	–	2.5	63%
6	–	–	166	–	3.0	50%
8	–	–	–	149	3.3	41%

Source: Author(s).

TABLE V.
MEASURES BY ITERATION OF THE PARALLEL PROGRAM WITH OMP FOR IN THE CLUSTER WITH $N = 100$ AND $T1 = 2011$, THE MEASURES OF TIME IN SECONDS.

p	t_2	t_4	t_6	t_8	S_p	E_f
1	–	–	–	–	–	–
2	1610	–	–	–	1.3	63%
4	–	1410	–	–	1.4	36%
6	–	–	1347	–	1.5	25%
8	–	–	–	1309	1.5	19%

Source: Author(s).

TABLE VI.
TOTAL TIME GAIN IN SECONDS WHEN PARALLELING WITH OMP.

n	Δt_{12}	Δt_{14}	Δt_{16}	Δt_{18}
10	40.1	56.1	64.4	64.3
20	80.1	120.9	128.8	137
50	199.7	296.1	327.9	345.1
100	401.1	601.5	664	702

Source: Author(s).

For the execution of the program, it is necessary to add the armadillo module, in the red box is marked the identifier of the assigned job (JOBID) after sending the program, to see the status of the run the following command is used:

$ squeue -l

The yellow box indicates the corresponding job, and it can be observed that it is working on the node 2 of the cluster with a single node (which is due to the job configuration), this node according to the cluster description has 8 processors.

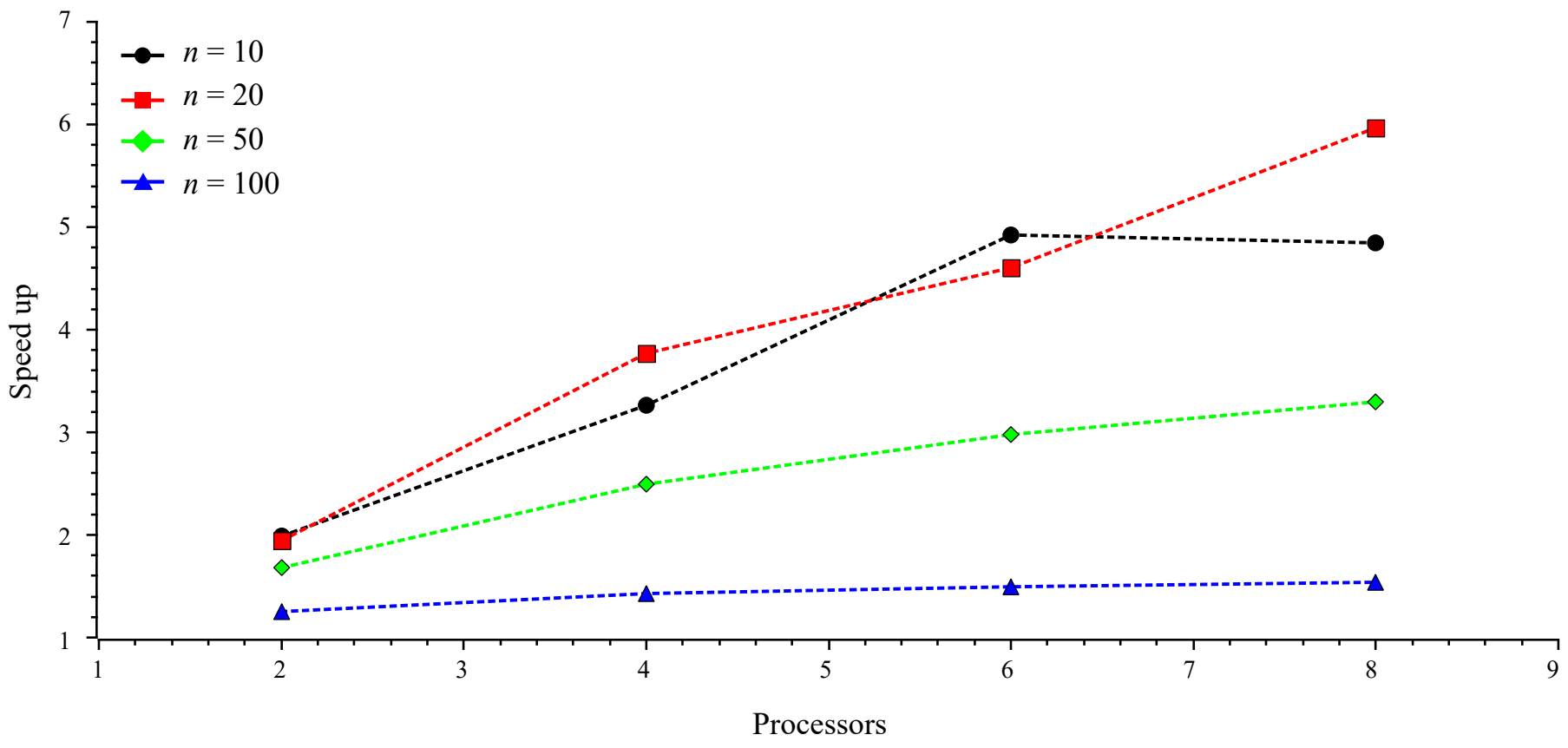

Fig. 5. Speedup for the final phase update problem paralleled with loops for up to $p = 8$ processors.
Source: Author(s).

The Fig. 5 and Fig. 6 represent the speedup and efficiency achieved in the cluster, respectively. The time measurements taken in the cluster were superior to those taken locally, this is due to the use of the armadillo library, since the libraries in the cluster are usually not installed at the system level, because sometimes several versions are used simultaneously, for this reason, when compiling it is necessary to specify additional directories for the compiler to find the .h, .a y/o .so.

The results obtained when increasing several processors present significant improvements when the dimensions n is relatively low. By increasing the dimensions of the model (looking for more precision), although these improvements continue to be present in the total saving of calculation time, the efficiency of the parallel architecture decreases. These results are mainly due to the portion of code that is paralleled within the program: since this portion does not constitute a large amount of code or embrace the largest number of calculations, the improvements in time are due only to what can provide the parallelized part.

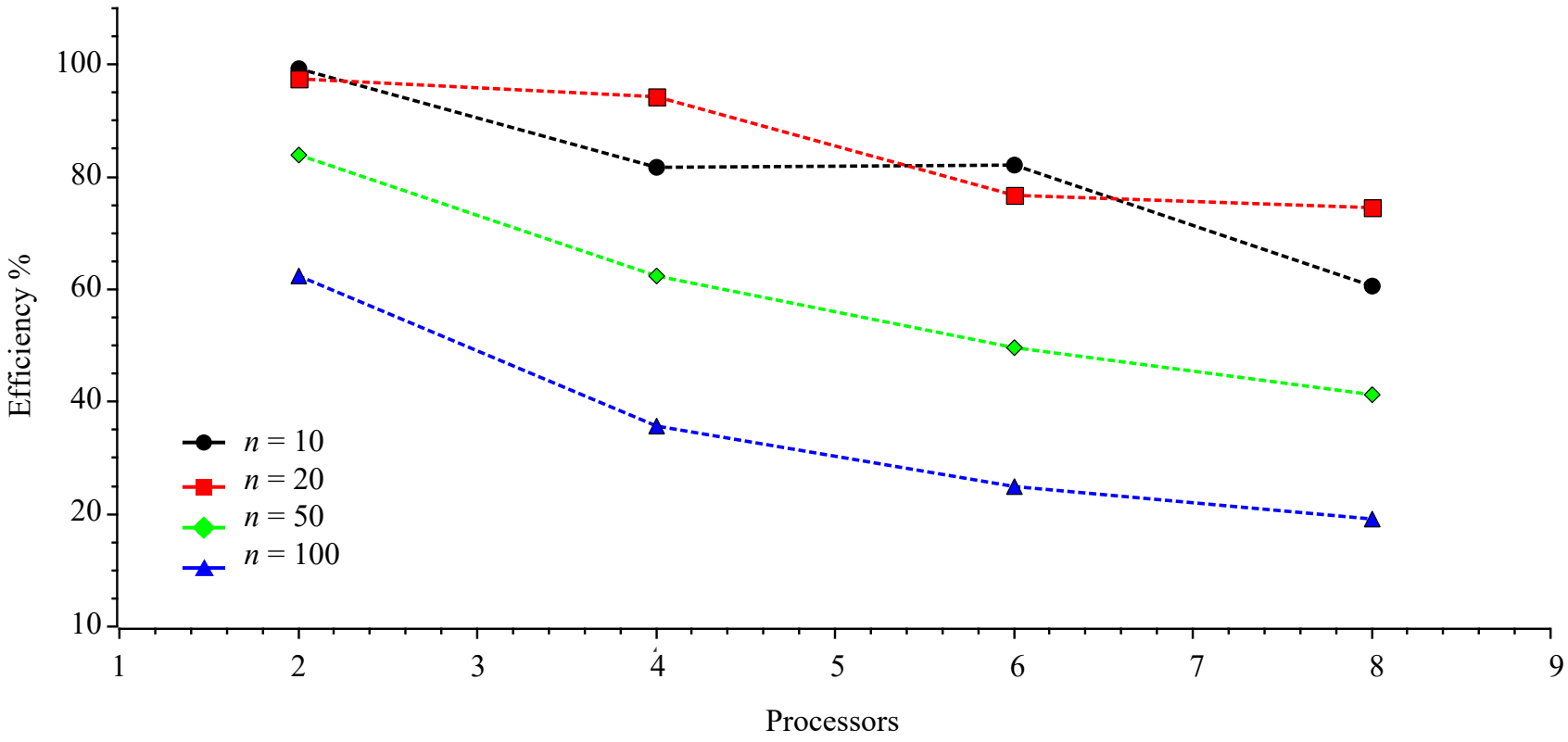

Fig. 6. Efficiency of the parallel architecture for the final phase update problem parallelized with loops for up to $p = 8$ processors.
Source: Author(s).

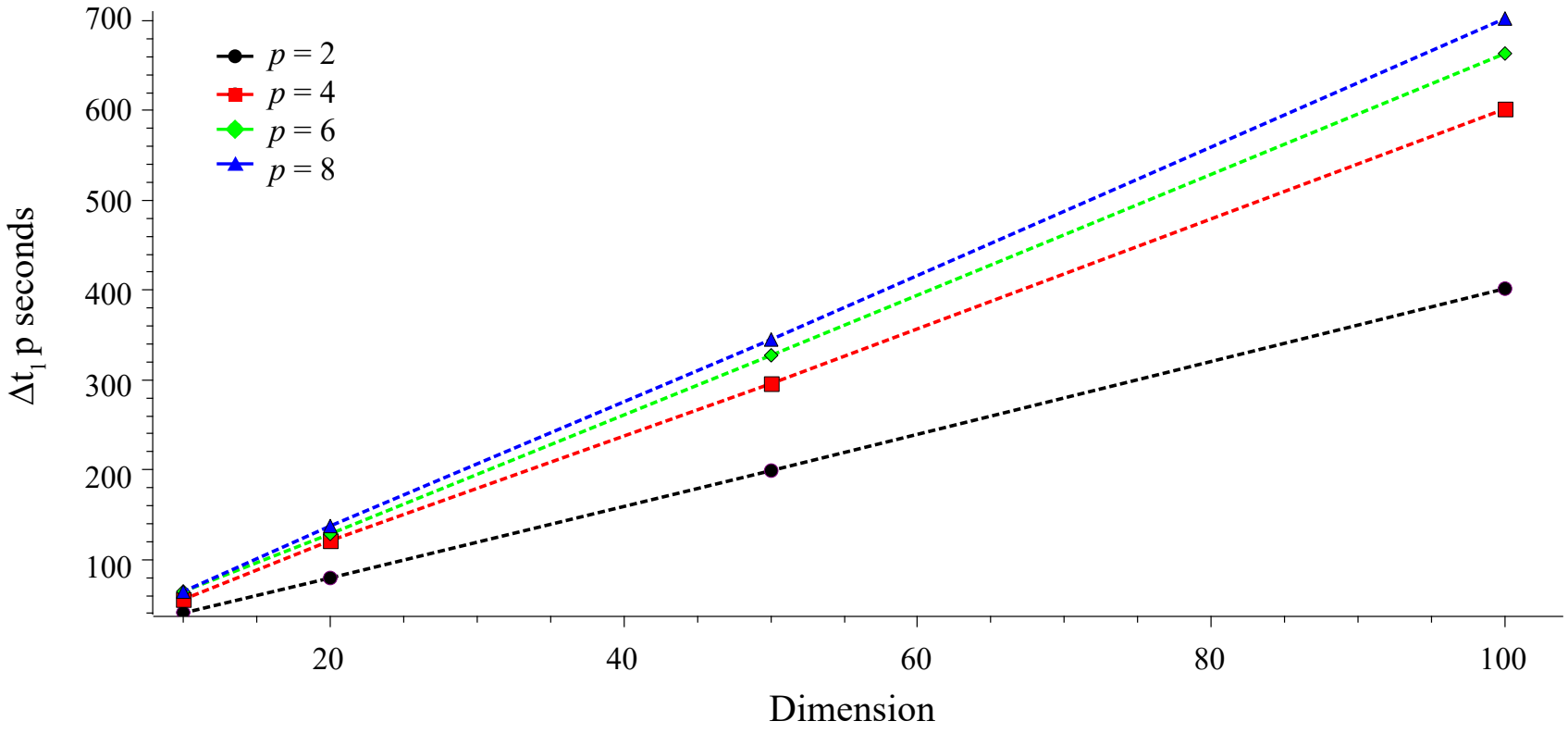

Fig. 7. Total gain of calculation time due to the progressive increase in processors.
Source: Author(s).

VI. Conclusions and Perspectives

For the problem of updating the Final Phase values studied [19], [20], explored and used OpenMP with a shared memory configuration, showing two strategies to parallelize the problem: (*i*) through the distribution by means of sections of the derivatives of cost with respect to the final values ρ, μ and the switching times τ_s, and (*ii*) through the parallelization of the loops involved in the sweeps of the final phase values ρi, μj (36), (37).

It was evidenced in section V-A2 that for models in which the fraction of program that can be paralleled does not constitute a major part of the total problem, the improvements that can be obtained due to working with parallel computing are limited, and it is not possible to obtain better results even if they increase and use gradually more processors in the cluster.

When working with programs with different characteristics, the scalability plays a fundamental role, therefore the possibility of making use of the cluster for the solution of the LQR problem with final penalty in which several periods of saturation are presented, also allowing the distribution of calculation in the same ones, which would imply a greater computational load. Also, for the treatment of non-linear models [30], the use of parallel computing is more appropriate, since it can reduce processing times when simulating states and costates trajectories.

References

[1] V. E. Sonzogni, A. M. Yommi, N. M. Nigro & M. A. Storti, "A parallel finite element program on a beowulf cluster," *Adv Eng Softw*, vol. 33, no. 7, pp. 427–443, Jul. 2002. https://doi.org/10.1016/S0965-9978(02)00059-5

[2] J. D. Hennessy & D. A. Patterson, *Arquitectura de computadores: Un enfoque cuantitativo*. MD, Esp.: McGraw-Hill, 1993.

[3] M. Tokhi, M. A. Hossain & M. Shaheed, *Parallel Computing for Real-Time Signal Processing and Control*. London, UK: Springer, 2003.

[4] A. S. Tanenbaum, Sistemas operativos modernos. *3rd edición*. MX, D.F., MX: Pearson Educación, 2009.

[5] P. Pardalos & R. Pytlak, *Conjugate Gradient Algorithms In Nonconvex Optimization*. NY, USA: Springer, 2008.

[6] A. A. Agrachev & Y. L. Sachkov, *Control Theory from the Geometric Viewpoint*. Bln-HDB: Springer-Verlag, 2004.

[7] M. Athans & P. Falb, *Optimal Control: An Introduction to the Theory and Its Applications*. NY, USA: Dover, 2006.

[8] V. Costanza & P. S. Rivadeneira, *Enfoque Hamiltoniano al control optimo de sistemas dinámicos*. SAANZ, DE: OmniScriptum, 2014.

[9] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze & E. F. Mishchenko, *The Mathematical Theory of Optimal Processes*. NY, USA: Macmillan, 1964.

[10] J. L. Troutman, *Variational Calculus and Optimal Control*. NY, USA: Springer, 1996.

[11] V. Costanza & C. E. Neuman, "Partial differential equations for missing boundary conditions in the linear-quadratic optimal control problems," *Lat Am Appl Res*, vol. 39, no. 3, pp. 207–212, Dec. 2009. Available: http://hdl.handle.net/11336/17096

[12] E. D. Sontag, *Mathematical Control Theory*. NY, USA: Springer, 1998.

[13] V. Costanza & C. E. Neuman, "Optimal control of nonlinear chemical reactors via an initial-value hamiltonian problem," *Optim Control Appl Methods*, vol. 27, no. 1, pp. 41–60, Jan. 2006. http://dx.doi.org/10.1002/oca.772

[14] A. Kojima & M. Morari, "LQ control for constrained continuous-time systems," *Automatica*, vol. 40, no. 7, pp. 1143–1155, Jul. 2004. https://doi.org/10.1016/j.automatica.2004.02.007

[15] S. J. Qin & T. A. Badgwell, "A survey of industrial model predictive control technology," *Control Eng Pract*, vol. 11, no. 7, pp. 733–764, Jul. 2003. https://doi.org/10.1016/S0967-0661(02)00186-7

[16] O. J. Rojas, G. C. Goodwin, M. M. Seron & A. Feuer, "An svd based´ strategy for receding horizon control of input constrained linear systems," *Int J Robust Nonlin*, vol. 14, no. 13-14, pp. 1207–1226, May. 2004. https://doi.org/10.1002/rnc.940

[17] J. L. Speyer & D. H. Jacobson, *Primer on Optimal Control Theory*. Phila, USA: SIAM Books, 2010.

[18] V. Costanza & P. S. Rivadeneira, "Optimal satured feedback laws for LQR problems with bounded controls," *Comput Appl Math*, vol. 32, no. 2, pp. 355–371, Mar. 2013. https://doi.org/10.1007/s40314-013-0025-7

[19] V. Costanza, P. S. Rivadeneira & J. A. Gómez, "An efficient cost reduction procedure for bounded-control LQR problems," *Comput Appl Math*, vol. 37, no. 2, pp. 1175–1196, Oct. 2016. https://doi.org/10.1007/s40314-016-0393-x

[20] V. Costanza, P. S. Rivadeneira & J. A. Gómez, "Numerical treatment of the bounded-control lqr problem by updating the final phase value," *IEEE Lat Ame T*, vol. 14, no. 6, pp. 2687–2692, Jun. 2016. https://doi.org/10.1109/TLA.2016.7555239

[21] V. Costanza & P. S. Rivadeneira, "Online suboptimal control of linearized models," *Syst Sci Control Eng*, vol. 2, no. 1, pp. 379–388, Dec. 2014. https://doi.org/10.1080/21642583.2014.913215

[22] E. Bramanti, M. Bramanti, P. Stiavetti & E. Benedetti, "A frequency deconvolution procedure using a conjugate gradient minimization method with suitable constraints," *J Chemom*, vol. 8, no. 6, pp. 409–421, Dec. 1994. https://doi.org/10.1002/cem.1180080606

[23] R. Fletcher & C. M. Reeves, "Function minimization for conjugate gradients," *Computer J*, vol. 7, no. 2, pp. 149–154, Jan. 1964. https://doi.org/10.1093/comjnl/7.2.149

[24] A. V. Rao, D. A. Benson, G. T. Huntington, C. Francolin, C. L. Darby & M. A. Patterson, "User's manual for GPOPS: A matlab package for dynamic optimization using the gauss pseudospectral method," UF, GVL; USA, *Tech. Rep.*, Aug. 2008.

[25] P. Bernhard, "Introduccion a la teoría de control Optimo," Inst. Mat. Beppo Levi, ROS, AR, *Tech. Rep., Cuaderno No. 4*, 1972.

[26] V. Costanza, P. S. Rivadeneira & R. D. Spies, "Equations for the missing boundary values in the hamiltonian formulation of optimal control problems," *J Optim Theory and Appl*, vol. 149, no. 1, pp. 26–46, Jan. 2011. https://doi.org/10.1007/s10957-010-9773-3

[27] G. C. Goodwin, S. F. Graebe & M. E. Salgado, Control system design, *vol. 240*. NJ, USA: Prentice Hall, 2001.

[28] W. S. Levine, *The control handbook*. BOCCA, USA: CRC Press, 1996.
[29] J.-J. E. Slotine & W. Li, Applied nonlinear control, *vol. 199, no. 1*. NJ, ENGI: Prentice-Hall, 1991.
[30] V. Costanza & P. S. Rivadeneira, "Partially-Regular Bounded-Control Problems for Nonlinear Systems," in Conf. Proc. *XXIV Congreso Argentino de Control Automático*, AADECA, BA, AR, 27-29 Oct. 2014.
[31] J. A. Gómez, P. S. Rivadeneira & V. Costanza, "A cost reduction procedure for control-restricted nonlinear systems," *IREACO*, vol. 10, no. 6, pp. 1–24, Nov. 2017. https://doi.org/10.15866/ireaco.v10i6.13820
[32] B. Chapman, G. Jost & R. Van Der Pas, *Using OpenMP*. CAM, MA, USA: MIT Press, 2008.
[33] V. Dhamo & F. Troltzsch, "Some aspects of reachability for parabolic¨ boundary control problems with control constraints," *Comput Optim Appl*, vol. 50, no. 1, pp. 75–110, Oct. 2008. https://doi.org/10.1007/s10589-009-9310-1
[34] G. Hearns & M. J. Grimble, "Temperature control in transport delay systems," in Proc. *2010 American Control Conference*, IEEE, Baltimore, MD, USA, 30 Jun-2 Jul. 2010, pp. 6089–6094. https://doi.org/10.1109/ACC.2010.5531540
[35] C. Sanderson & R. Curtin, "Armadillo: a template-based c++ library for linear algebra," *J Open Source Softw*, vol. 1, no. 2, pp. 1–2, Jun. 2016. https://doi.org/10.21105/joss.00026