Received: 21 February 2020 | Revised: 19 March 2022 | Accepted: 27 June 2022

DOI: 10.1002/oca.2927

RESEARCH ARTICLE

Parallel solution of optimal control problems using the graphics processing unit

Chaoyi Yang | **Brian C. Fabien**

Mechanical Engineering, University of Washington, Seattle, Washington, USA

Correspondence
Chaoyi Yang, Mechanical Engineering, University of Washington, Seattle, Washington 98195, USA
Email: yangc22@uw.edu

Funding information
National Science Foundation, Grant/Award Number: 1542228

Abstract

This paper presents an indirect method for solving optimal control problems (OCPs) using graphics processing unit (GPU). The OCPs considered here include control variable inequality constraints (CVICs), state variable inequality constraints (SVICs), and parameters. The necessary conditions of the minimum for the OCPs are written as a boundary value problem with index-1 differential algebraic equations (BVP-DAEs). The complementarity conditions associated with those inequality constraints are approximated using Kanzow's smoothed Fisher-Burmeister formula. The numerical solution for solving the BVP-DAEs is based on the multiple shooting technique and the DAEs are solved using a single step linearly implicit Runge–Kutta (Rosenbrock–Wanner, ROW) method. A Newton's continuation method is performed to solve the BVP-DAEs system and the descent direction is found by solving a sparse bordered almost block diagonal (BABD) linear system with a structured orthogonal factorization algorithm. Parallel computing techniques are used to accelerate the code which is implemented using Python and CUDA on GPU. Numerical examples are presented to illustrate the efficiency of the code. The GPU based parallel implementation is shown to be significantly faster than the implementation using central processing unit (CPU) alone.

KEYWORDS

boundary value problem, CUDA, GPU based parallel programming, index-1 differential algebraic equation, multiple shooting method, optimal control, slacked unconstrained penalty function method

1 | INTRODUCTION

Optimal control problems (OCPs) deal with the problems of finding a control law for a given system such that a certain optimality criterion is achieved. The favored approaches for solving optimal control problems are that of the direct method and the indirect method (Bock and Plitt,[1] Fabien,[2] and Fabien[3]). In an indirect method, the calculus of variations is employed to obtain the first-order optimality conditions. These conditions result in a boundary value problem involving differential-algebraic equations (BVP-DAEs) and the problem is then solved numerically (Gerdts,[4] Fabien,[5] and Fabien[6]). The problem solving process involves a large amount of computation and in applications requiring real-time optimal control the problem can benefit from the faster execution.

In this article, we present an indirect method for solving OCPs with control variable inequality constraints (CVICs), state variable inequality constraints (SVICs) and parameters. The paper aims to take advantage of the computing capability of the Graphical Processing Units (GPU) as the algorithm presented is highly suitable to be executed in parallel on the large number of GPU cores. Using the GPU to accelerate the computation within a single compute node can be an efficient way of improving performance of the algorithm.

Section 2 presents the problem statement and the essential assumptions. Here, the original OCP with constraints is modified using the slacked unconstrained penalty function method. The resulting first-order necessary conditions are written in the form of a two-point boundary value problem involving differential algebraic equations that are shown to be index-1. Moreover, the complementarity conditions in the differential algebraic equations are approximated using Kanzow's smoothed Fischer–Burmeister formula (see Kanzow[7] and Fabien[8]).

Section 3 presents a multiple shooting method to solve the BVP-DAEs. This multiple shooting method leads to a system of residual equations that determines the unknowns on a fixed mesh in the time interval of interest. The residual equation is solved using a damped Newton's continuation method. The evaluation of the multiple shooting residual equation, and its Jacobian are integrated together using a single step linearly implicit Runge–Kutta (Rosenbrock–Wanner, ROW) method (Hairer and Wanner[9]). The computation of the search direction for solving the residual equation requires the solution of a sparse bordered almost block diagonal (BABD) linear system (see Amodio et al.,[10] and Fabien[6]).

Section 4 provides the Python and CUDA (Storti and Yurtoglu,[11] Nickolls et al.,[12] and Luebke[13]) implementation of the algorithm. This section also presents examples to validate the accuracy and efficiency of the proposed approach. The paper concludes in Section 5 with a summary of the main contributions.

Many more details of the algorithm can be found in previous publications; in particular, Fabien[6] gives an extensive discussion of the multiple shooting algorithm, along with an ANSI C(99) parallel implementation of the software with nontrival examples, while Fabien[14] discusses more details of the slacked unconstrained penalty function method. Yang and Fabien[15] presents a GPU based parallel collocation method to solve constrained optimal control problems.

2 | OPTIMAL CONTROL PROBLEM

This paper develops an algorithm to find the state vector $x(t) \in \mathbb{R}^{n_x}$, the control vector $u(t) \in \mathbb{R}^{n_u}$, and the parameter vector $w \in \mathbb{R}^{n_w}$, which minimizes the cost functional of the optimal control problem (OCP)

$$J(x,u,w) = \phi(x(t_\mathrm{f}),w) + \int_{t_\mathrm{i}}^{t_\mathrm{f}} L(x(t),u(t),w)dt, \tag{1}$$

subject to the constraints

$$\dot{x}(t) = f(x(t),u(t),w), \quad t \in [t_\mathrm{i}, t_\mathrm{f}], \tag{2}$$

$$0 = \Gamma(x(t_\mathrm{i}),w), \tag{3}$$

$$0 \geq d_i(x(t),u(t),w), \quad i = 1,2, \dots , n_d, \quad t \in [t_\mathrm{i}, t_\mathrm{f}], \tag{4}$$

$$0 \geq s_j(x(t)), \quad j = 1,2, \dots , n_s, \quad t \in [t_\mathrm{i}, t_\mathrm{f}], \tag{5}$$

$$0 = \Psi(x(t_\mathrm{f}),w). \tag{6}$$

The cost functional $J(x,u,w)$ is made up of a scalar terminal penalty term $\phi(x(t_\mathrm{f}),w)$, and a scalar integral term with integrand $L(x(t),u(t),w)$. The optimal solution must satisfy the differential equations (2) where $f(x(t),u(t),w) \in \mathbb{R}^{n_x}$, the initial time constraints (3) where $\Gamma(x(t_\mathrm{i}),w) \in \mathbb{R}^{n_\Gamma}$, the mixed control state parameter inequality constraints (4), the state variable inequality constraints (5), and the final time constraints (6) where $\Psi(x(t_\mathrm{f}),w) \in \mathbb{R}^{n_\Psi}$. The initial and final time of the problem considered are known and fixed. In addition, the dynamic system equations are autonomous. Note that a problem with unknown final time can be treated as a problem with fixed final time and a parameter representing the unknown final time. Also, nonautomous systems can be written as autonomous systems by using a new state variable to represent the time variable.

The numerical solution developed in this article relies on the following assumptions.

Assumption 1 (Existence of a solution). There exist a state vector $x(t)$, a control vector $u(t)$ and a parameter vector w that solve the problem defined by (1)–(6).

Assumption 2 (Smoothness of functions). The functions $\phi, L, f, \Gamma, d_i, i = 1, 2, \dots, n_d, s_j, j = 1, 2, \dots, n_s$ and Ψ are at least twice differentiable with respect to their arguments.

The ith control state parameter inequality constraint is said to be active if $d_i(x, u, w) = 0$, while the constraint is said to be inactive if $d_i(x, u, w) < 0$. Let the index set of the active inequality constraints be denoted as $\mathbf{A}(x, u, w) = \{i \mid d_i(x, u, w) = 0\}$, with the cardinality of $\mathbf{A}(x, u, w)$ equal to $\overline{n}_d$. Here, $D_u d_{\mathbf{A}}(x, u, w)$ is the n_u by $\overline{n}_d$ matrix whose columns are $\partial d_i(x, u, w)/\partial u,\ i \in \mathbf{A}(x, u, w)$.

Assumption 3 (Constraint qualification). The optimal control of the problem is such that the gradients of the active control state parameter inequality constraints are linearly independent all the time, that is,

$$\operatorname{rank}\left[D_u d_{\mathbf{A}}(x, u, w)\right] = \overline{n}_d.$$

2.1 | Transformation of the problem

In order to obtain an approximate solution to OCP (1)–(6), we first consider the following relaxed OCP. Find $x(t) \in \mathbb{R}^{n_x}$, $u(t) \in \mathbb{R}^{n_u}$, $v(t) \in \mathbb{R}^{n_d}$, $\sigma(t) \in \mathbb{R}^{n_s}$, and $w \in \mathbb{R}^{n_w}$ that minimizes the cost functional

$$\begin{aligned}\hat{J}(x, u, w, v, \sigma; \alpha) = \phi(x(t_\mathrm{f}), w) + \int_{t_\mathrm{i}}^{t_\mathrm{f}} & L(x(t), u(t), w) \\ & + \frac{1}{2\alpha}\sum_{i=1}^{n_d}(d_i(x(t), u(t), w) + v_i(t))^2 \\ & + \frac{1}{2\alpha}\sum_{j=1}^{n_s}(s_j(x(t)) + \sigma_j(t))^2 \\ & + \frac{\alpha}{2}u(t)^T u(t) dt \end{aligned} \tag{7}$$

subject to the constraints

$$\dot{x}(t) = f(x(t), u(t), w), \quad t \in [t_\mathrm{i}, t_\mathrm{f}], \tag{8}$$

$$0 = \Gamma(x(t_\mathrm{i}), w), \tag{9}$$

$$0 \le v_i(t), \quad i = 1, 2, \dots, n_d, \quad t \in [t_\mathrm{i}, t_\mathrm{f}], \tag{10}$$

$$0 \le \sigma_j(t), \quad j = 1, 2, \dots, n_s, \quad t \in [t_\mathrm{i}, t_\mathrm{f}], \tag{11}$$

$$0 = \Psi(x(t_\mathrm{f}), w). \tag{12}$$

In this problem formulation $\alpha > 0$ is a penalty parameter. Here, the CVICs and SVICs are treated as penalty terms using the non-negative slack variables $v(t)$ and $\sigma(t)$, respectively. These slack variables are considered to be control inputs in this formulation. Also, the term $\frac{\alpha}{2}u(t)^T u(t)$ is added to ensure that the second-order condition (Assumption 4 below) is satisfied when $\alpha > 0$ is sufficiently large, which is important in proving Theorem 2. Note that this formulation is used to extend the results given in Fabien.[8]

The usefulness of this slack unconstrained transformation is demonstrated in the following theorem.

Theorem 1. *Assume that the feasible regions for both the original optimal control problem (1)–(6) and the transformed optimal control problem (7)–(12) are compact and nonempty. Let $\{\alpha^{(k)}\}$ be an infinite sequence of positive numbers such that $\alpha^{(k)} > \alpha^{(k+1)} > 0$ and $\lim_{k\to\infty}\alpha^{(k)} = 0$. Let $(x^{(k)}, u^{(k)}, w^{(k)}, v^{(k)}, \sigma^{(k)})$ represent the bounded solution to the transformed optimal control problem with the penalty parameter $\alpha = \alpha^{(k)}$. Then, $\lim_{k\to\infty}\hat{J}(x^{(k)}, u^{(k)}, w^{(k)}, v^{(k)}, \sigma^{(k)}; \alpha^{(k)}) = J(x^*, u^*, w^*)$, where the triple (x^*, u^*, w^*) solves the optimal solution for the original optimal control problem.*

Proof. To prove this theorem we can restate the original and the transformed optimal control problems as follows. First, define the feasible set of the original problem (1)–(6) as $\Omega(x, u, w) = \{x(t), u(t), w | \dot{x}(t) - f(x, u, w) = 0,\ \Gamma(x(t_\mathrm{i}), w) = 0,\ \Psi(x(t_\mathrm{f}), w) = 0,\ d_i(x, u, w) \le 0,\ i = 1, 2, \dots, n_d,\ s_j(x) \le 0,\ j = 1, 2, \dots, n_s\}$. Then, the original problem becomes

$$\min_{(x(t),u(t),w)\in\Omega} J(x, u, w)$$

with $J(x^*, u^*, w^*) > -\infty$. Define $v_i^*(t) = -d_i(x^*(t), u^*(t), w^*)$, $i = 1, 2, \dots, n_d$ and $\sigma_j^*(t) = -s_j(x^*(t))$, $j = 1, 2, \dots, n_s$. Using these definitions, it can be shown that $D^* = \frac{1}{2}\int_{t_\mathrm{i}}^{t_\mathrm{f}}\Sigma_{i=1}^{n_d}(d_i(x^*(t), u^*(t), w^*) + v_i^*(t))^2 dt = 0$ and $S^* = \frac{1}{2}\int_{t_\mathrm{i}}^{t_\mathrm{f}}\Sigma_{j=1}^{n_s}(s_i(x^*(t)) + \sigma_j^*(t))^2 dt = 0$. Moreover, define $U^* = \frac{1}{2}\int_{t_\mathrm{i}}^{t_\mathrm{f}}((u^*(t))^T u^*(t))dt < \infty$ since $u^*(t)$ is bounded.

Define the feasible set of the transformed problem (7)–(12) as $\Lambda(x, u, w, v, \sigma) = \{x, u, w, v, \sigma | \dot{x}(t) - f(x, u, w) = 0,\ \Gamma(x(t_\mathrm{i}), w) = 0,\ \Psi(x(t_\mathrm{f}), w) = 0,\ v_i(t) \ge 0,\ i = 1, 2, \dots, n_d,\ \sigma_j(t) \ge 0,\ j = 1, 2, \dots, n_s\}$. Then, the transformed optimal control problem becomes

$$\min_{(x(t),u(t),w,v(t),\sigma(t))\in\Lambda} \hat{J}(x, u, w, v, \sigma; \alpha).$$

For every $\alpha^{(k)}$, denote $(x^{(k)}(t), u^{(k)}(t), w^{(k)}, v^{(k)}(t), \sigma^{(k)}(t))$ as the bounded optimal solution to the problem

$$\min_{(x(t),u(t),w,v(t),\sigma(t))\in\Lambda} \hat{J}(x, u, w, v, \sigma; \alpha^{(k)}).$$

Define $D^{(k)}(x^{(k)}(t), u^{(k)}(t), w^{(k)}, v_i^{(k)}(t)) = \frac{1}{2}\int_{t_\mathrm{i}}^{t_\mathrm{f}}\sum_{i=1}^{n_d}(d_i(x^{(k)}(t), u^{(k)}(t), w^{(k)}) + v_i^{(k)}(t))^2\, dt$ and $S^{(k)}(x^{(k)}(t), \sigma_j^{(k)}(t)) = \frac{1}{2}\int_{t_\mathrm{i}}^{t_\mathrm{f}}\sum_{j=1}^{n_s}(s_i(x^{(k)}(t)) + \sigma_j^{(k)}(t))^2\, dt$. It can be concluded that for all k

$$\hat{J}(x^{(k)}, u^{(k)}, w^{(k)}, v^{(k)}, \sigma^{(k)}; \alpha^{(k)}) \le \hat{J}(x^*, u^*, w^*, v^*, \sigma^*; \alpha^{(k)})$$
$$\hat{J}(x^{(k)}, u^{(k)}, w^{(k)}, v^{(k)}, \sigma^{(k)}; \alpha^{(k)}) \tag{13}$$

$$\begin{aligned} &\le J(x^*, u^*, w^*) + \frac{1}{\alpha^{(k)}}D^* + \frac{1}{\alpha^{(k)}}S^* + \alpha^{(k)}U^* \\ &= J(x^*, u^*, w^*) + \alpha^{(k)}U^*. \end{aligned} \tag{14}$$

Since Equation (14) holds for every $\alpha^{(k)}$, we must have $\lim_{k\to\infty} D^{(k)}(x^{(k)}(t), u^{(k)}(t), w^{(k)}, v_i^{(k)}(t)) = 0$ and $\lim_{k\to\infty} S^{(k)}(x^{(k)}(t), \sigma_j^{(k)}(t)) = 0$. Otherwise, the left hand side of the equation becomes unbounded while the right hand side approaches $J(x^*, u^*, w^*)$, which contradicts the equation.

Let $(\overline{x}, \overline{u}, \overline{w}, \overline{v}, \overline{\sigma})$ be the limit of the sequence $(x^{(k)}, u^{(k)}, w^{(k)}, v^{(k)}, \sigma^{(k)})$. Then, the last inequality Equation (14) indicates that

$$\lim_{k\to\infty}\hat{J}(x^{(k)}, u^{(k)}, w^{(k)}, v^{(k)}, \sigma^{(k)}; \alpha^{(k)}) = J(\overline{x}, \overline{u}, \overline{w}) \le J(x^*, u^*, w^*). \tag{15}$$

Also, because $(d_i(\overline{x}, \overline{u}, \overline{w}) + \overline{v}_i) = 0,\ i = 1, 2, \dots, n_d$ and $(s_j(\overline{x}) + \overline{\sigma}_j) = 0,\ j = 1, 2, \dots, n_s$, we get that $(\overline{x}, \overline{u}, \overline{w}) \in \Omega$. Since (x^*, u^*, w^*) minimizes $J(x, u, w)$ on Ω, we get

$$J^* = J(x^*, u^*, w^*) \le J(\overline{x}, \overline{u}, \overline{w}). \tag{16}$$

Combining Equations (15) and (16), we have $J(\overline{x}, \overline{u}, \overline{w}) = J(x^*, u^*, w^*)$, which proves the theorem. ■

2.2 Necessary conditions

The necessary conditions for (x, u, w, v, σ) to be a minimum of the OCP (7)–(12) can be derived using the calculus of variations and the Lagrange multiplier rule (see Agrawal and Fabien,[16] Fabien,[14] and Hartl et al.[17]).

To state the necessary conditions, define the scalar Hamiltonian function as

$$\overline{H} = L(x,u,w) + \frac{1}{2\alpha}\sum_{i=1}^{n_d}(d_i(x,u,w)+v_i)^2 + \frac{1}{2\alpha}\sum_{j=1}^{n_s}(s_j(x)+\sigma_j)^2 + \lambda^T f(x,u,w) - \mu^T v - \xi^T\sigma + \frac{\alpha}{2}u(t)^T u(t),$$

where $\lambda(t) \in \mathbb{R}^{n_x}$ are the costate variables, $\mu(t) \in \mathbb{R}^{n_d}$ are the Lagrange multipliers associated with the non-negative slack variables $v(t)$, and $\xi(t) \in \mathbb{R}^{n_s}$ are the Lagrange multipliers associated with the non-negative slack variables $\sigma(t)$.

As we defined $v(t)$ and $\sigma(t)$ as the control inputs, if (x, u, w, v, σ) represents a local minimum of the OCP, then it is necessary that the following conditions be satisfied.

$$\dot{x} = \frac{\partial \overline{H}}{\partial \lambda}, \tag{17}$$

$$\dot{\lambda} = -\frac{\partial \overline{H}}{\partial x}, \tag{18}$$

$$\dot{\gamma} = \frac{\partial \overline{H}}{\partial w}, \tag{19}$$

$$0 = \frac{\partial \overline{H}}{\partial u}, \tag{20}$$

$$0 = \frac{\partial \overline{H}}{\partial v_i} = \frac{1}{\alpha}(d_i + v_i) - \mu_i, \quad i = 1, 2, \dots, n_d, \tag{21}$$

$$0 = \frac{\partial \overline{H}}{\partial \sigma_j} = \frac{1}{\alpha}(s_j + \sigma_j) - \xi_j, \quad j = 1, 2, \dots, n_s, \tag{22}$$

$$0 = \mu_i v_i, \quad \mu_i \geq 0, \quad v_i \geq 0, \quad i = 1, 2, \dots, n_d, \tag{23}$$

$$0 = \xi_j \sigma_j, \quad \xi_j \geq 0, \quad \sigma_j \geq 0, \quad j = 1, 2, \dots, n_s, \tag{24}$$

$$0 = \Gamma(x(t_\mathrm{i}), w), \tag{25}$$

$$0 = \lambda(t_\mathrm{i}) + D_x\Gamma(x(t_\mathrm{i}), w)K_\mathrm{i}, \tag{26}$$

$$0 = \gamma(t_\mathrm{i}) - D_w\Gamma(x(t_\mathrm{i}), w)K_\mathrm{i}, \tag{27}$$

$$0 = \Psi(x(t_\mathrm{f}), w), \tag{28}$$

$$0 = \lambda(t_\mathrm{f}) - \frac{\partial \phi}{\partial x} - D_x\Psi(x(t_\mathrm{f}), w)K_\mathrm{f}, \tag{29}$$

$$0 = \gamma(t_\mathrm{f}) + \frac{\partial \phi}{\partial w} + D_w\Psi(x(t_\mathrm{f}), w)K_\mathrm{f}. \tag{30}$$

Equations (21) and (22) can be further simplified as

$$0 = d_i(x,u,w) + v_i - \alpha\mu_i, \quad i = 1, 2, \dots, n_d, \tag{31}$$

$$0 = s_j(x) + \sigma_j - \alpha\xi_j, \quad j = 1, 2, \dots, n_s, \tag{32}$$

by multiplying the penalty parameter α on both sides.

2.3 | Transformation of the complementarity conditions

Fabien[8] approximated the complementarity conditions using the Kanzow's smoothed Fisher–Burmeister formula (see Kanzow[7]), $\psi : \mathbb{R}^2 \Rightarrow \mathbb{R}$, which is

$$\psi(a, b; \alpha) = a + b - \sqrt{a^2 + b^2 + 2\alpha}. \tag{33}$$

It is proved in Kanzow[7] that formula (33) satisfies the following properties which are critical for proving Theorem 2.

1. For any $\alpha > 0$, $\psi(a, b; \alpha) = 0 \Leftrightarrow ab = \alpha,\ a > 0,\ b > 0.$
2. For all $(a, b) \in \mathbb{R}^2,\ \alpha > 0,$

$$0 < \frac{\partial\psi(a, b; \alpha)}{\partial a} < 2, \quad 0 < \frac{\partial\psi(a, b; \alpha)}{\partial b} < 2.$$

Therefore, the complementarity conditions (23)–(24) can be approximated by the smooth formula (33) and can be rewritten as

$$0 = \psi(\mu_i, \nu_i; \alpha), \quad i = 1, 2, \dots, n_d, \tag{34}$$

$$0 = \psi(\xi_j, \sigma_j; \alpha), \quad j = 1, 2, \dots, n_s. \tag{35}$$

Thus, equation (34) implies that

$$\mu_i \nu_i = \alpha, \quad \mu_i > 0, \quad \nu_i > 0, \quad i = 1, 2, \dots, n_d,$$

and equation (35) implies that

$$\xi_j \sigma_j = \alpha, \quad \xi_j > 0, \quad \sigma_j > 0, \quad j = 1, 2, \dots, n_s,$$

which are the same as the complementarity conditions (23) and (24) from the first-order necessary conditions when α approaches 0. The implementation of this paper uses the penalty parameter $\alpha^{(k)}$ as the parameter α here for every iteration.

2.4 | Boundary value problem

The necessary conditions can be written more compactly by separating the unknowns into: (i) differential variables; (ii) algebraic variables; and (iii) parameter variables. Specifically, define

$$y(t) = \begin{bmatrix} x(t) \\ \lambda(t) \\ \gamma(t) \end{bmatrix}, \quad z(t) = \begin{bmatrix} u(t) \\ \mu(t) \\ \xi(t) \\ \nu(t) \\ \sigma(t) \end{bmatrix}, \quad p = \begin{bmatrix} w \\ K_\mathrm{i} \\ K_\mathrm{f} \end{bmatrix}, \tag{36}$$

where $y(t) \in \mathbb{R}^{n_y}$ are the differential variables, $z(t) \in \mathbb{R}^{n_z}$ are the algebraic variables, and $p \in \mathbb{R}^{n_p}$ are the parameter variables, with $n_y = 2n_x + n_w$, $n_z = n_u + 2n_d + 2n_s$, and $n_p = n_w + n_\Gamma + n_\Psi$. The vector $\gamma(t) \in \mathbb{R}^{n_w}$ is used to write the stationary conditions associated with the parameters w as a differential equation instead of an integral equation. The Lagrange multipliers $K_\mathrm{i} \in \mathbb{R}^{n_\Gamma}$ are associated with the constraints at the initial time and the Lagrange multipliers $K_\mathrm{f} \in \mathbb{R}^{n_\Psi}$ are associated with the constraints at the final time. Moreover, $\mu(t) \in \mathbb{R}^{n_d}$, $\xi(t) \in \mathbb{R}^{n_s}$, $\nu(t) \in \mathbb{R}^{n_d}$, and $\sigma(t) \in \mathbb{R}^{n_s}$.

Using these definitions, the necessary conditions for optimality (17)–(30) can be written as the two-point BVP-DAEs

$$\dot{y} = h(y(t), z(t), p), \tag{37}$$

$$0 = g(y(t), z(t), p, \alpha), \tag{38}$$

$$0 = r(y(t_\mathrm{i}), y(t_\mathrm{f}), p). \tag{39}$$

Here, (37) defines a set of differential equations, (38) defines a set of algebraic equations, and (39) defines a set of boundary conditions. Moreover

$$h(y(t), z(t), p) = \begin{bmatrix} \partial\overline{H}/\partial\lambda \\ -\partial\overline{H}/\partial x \\ \partial\overline{H}/\partial w \end{bmatrix} \in \mathbb{R}^{n_y}, \tag{40}$$

$$g(y(t), z(t), p, \alpha) = \begin{bmatrix} \partial\overline{H}/\partial u \\ d(x,u,w) + \mathcal{N}e - \alpha\mathcal{M}e \\ s(x) + \mathcal{S}e - \alpha\mathcal{E}e \\ \psi_d(\mu, \nu; \alpha) \\ \psi_s(\xi, \sigma; \alpha) \end{bmatrix} \in \mathbb{R}^{n_z}, \tag{41}$$

$$r(y(t_\mathrm{i}), y(t_\mathrm{f}), p) = \begin{bmatrix} \Gamma(x(t_\mathrm{i}), w) \\ \lambda(t_\mathrm{i}) + D_x\Gamma(x(t_\mathrm{i}), w)K_\mathrm{i} \\ \gamma(t_\mathrm{i}) - D_w\Gamma(x(t_\mathrm{i}), w)K_\mathrm{i} \\ \Psi(x(t_\mathrm{f}), w) \\ \lambda(t_\mathrm{f}) - \frac{\partial\phi}{\partial x} - D_x\Psi(x(t_\mathrm{f}), w)K_\mathrm{f} \\ \gamma(t_\mathrm{f}) + \frac{\partial\phi}{\partial w} + D_w\Psi(x(t_\mathrm{f}), w)K_\mathrm{f} \end{bmatrix} \in \mathbb{R}^{n_y+n_p}, \tag{42}$$

$d(x,u,w) = [d_1(x,u,w), d_2(x,u,w), \dots, d_{n_d}(x,u,w)]^T$, $s(x) = [s_1(x), s_2(x), \dots, s_{n_s}(x)]^T$, $\mathcal{N} = \mathrm{diag}(\nu_1, \nu_2, \dots, \nu_{n_d})$, $\mathcal{M} = \mathrm{diag}(\mu_1, \mu_2, \dots, \mu_{n_d})$, $\mathcal{S} = \mathrm{diag}(\sigma_1, \sigma_2, \dots, \sigma_{n_s})$, $\mathcal{E} = \mathrm{diag}(\xi_1, \xi_2, \dots, \xi_{n_s})$, $\psi_d(\mu, \nu; \alpha) = [\psi(\mu_1, \nu_1; \alpha), \psi(\mu_2, \nu_2; \alpha), \dots, \psi(\mu_{n_d}, \nu_{n_d}; \alpha)]^T$, $\psi_s(\xi, \sigma; \alpha) = [\psi(\xi_1, \sigma_1; \alpha), \psi(\xi_2, \sigma_2; \alpha), \dots, \psi(\xi_{n_s}, \sigma_{n_s}; \alpha)]^T$, $D_x\Gamma = [\partial\Gamma_1/\partial x, \partial\Gamma_2/\partial x, \dots, \partial\Gamma_{n_\Gamma}/\partial x]^T$, $D_w\Gamma = [\partial\Gamma_1/\partial w, \partial\Gamma_2/\partial w, \dots, \partial\Gamma_{n_\Gamma}/\partial w]^T$, $D_x\Psi = [\partial\Psi_1/\partial x, \partial\Psi_2/\partial x, \dots, \partial\Psi_{n_\Psi}/\partial x]^T$, $D_w\Psi = [\partial\Psi_1/\partial w, \partial\Psi_2/\partial w, \dots, \partial\Psi_{n_\Psi}/\partial w]^T$, and $e = [1, 1, \dots, 1]^T$.

Next, it can be shown that the BVP-DAEs (37)–(39) are index-1 along the optimal trajectory. To do so we require the second order necessary condition be satisfied. Hence, we make the following assumption.

Assumption 4 (The second order necessary condition). The matrix $\overline{H}_{uu} = \partial^2\overline{H}/\partial u^2 \in \mathbb{R}^{n_u \times n_u}$ is positive definite along the optimal trajectory. That is, for each bounded vector function $\hat{u}(t) \in \mathbb{R}^{n_u}$, in the interval $[t_\mathrm{i}, t_\mathrm{f}]$, it is assumed that there is a constant $c > 0$, such that $\hat{u}(t)^T\overline{H}_{uu}\hat{u}(t) \geq c\|\hat{u}(t)\|^2$.

Theorem 2. *For $\alpha > 0$, if (x, u, w, ν, σ) solves the OCP (7)–(12), then there exists vector (y, z, p) that solves the BVP-DAEs (37)-(39) and $\partial g/\partial z$ is non-singular; that is, the BVP-DAEs are index-1.*

Proof. Equations (37)–(39) are the restatement of the necessary conditions (17)–(30). Therefore, the proof should show that the BVP-DAEs satisfy the index-1 condition which is $\partial g/\partial z$ is non-singular along the optimal trajectory for $\alpha > 0$. The proof is done via contradiction. Suppose that $\partial g/\partial z$ is singular, then there should exist $\Delta u \in \mathbb{R}^{n_u}$, $\Delta\mu \in \mathbb{R}^{n_d}$, $\Delta\xi \in \mathbb{R}^{n_s}$, $\Delta\nu \in \mathbb{R}^{n_d}$, and $\Delta\sigma \in \mathbb{R}^{n_s}$ not being all zero such that

$$\left[\frac{\partial g}{\partial z}\right]\begin{bmatrix}\Delta u\\ \Delta\mu\\ \Delta\xi\\ \Delta\nu\\ \Delta\sigma\end{bmatrix}=\begin{bmatrix}0\\0\\0\\0\\0\end{bmatrix}, \tag{43}$$

where

$$\left[\frac{\partial g}{\partial z}\right]=\begin{bmatrix}\overline{H}_{uu} & D_u d & 0 & 0 & 0\\ D_u d^T & -\alpha I & 0 & I & 0\\ 0 & 0 & -\alpha I & 0 & I\\ 0 & D_\mu\psi_d & 0 & D_\nu\psi_d & 0\\ 0 & 0 & D_\xi\psi_s & 0 & D_\sigma\psi_s\end{bmatrix}.$$

In this expression, $\overline{H}_{uu}=\partial^2\overline{H}/\partial u^2$, $D_u d=[\partial d_1/\partial u,\ \dots\ ,\partial d_{n_d}/\partial u]$, $D_\mu\psi_d=\mathrm{diag}(\partial\psi_{d_i}/\partial\mu_i)$, $D_\nu\psi_d=\mathrm{diag}(\partial\psi_{d_i}/\partial\nu_i)$, $i=1,2,\ \dots\ ,n_d$, $D_\xi\psi_s=\mathrm{diag}(\partial\psi_{s_j}/\partial\xi_j)$, $D_\sigma\psi_s=\mathrm{diag}(\partial\psi_{s_j}/\partial\sigma_j)$, $j=1,2,\ \dots\ ,n_s$. From Section 2.3, it is implied that for $\alpha>0$, $D_\mu\psi_d$, $D_\nu\psi_d$, $D_\xi\psi_s$, and $D_\sigma\psi_s$ are all positive definite matrices.

First it is easy to show that the third and fifth block rows in (43) yield $\Delta\xi=0$ and $\Delta\sigma=0$. The fourth row shows $\Delta\nu=-(D_\nu\psi_d)^{-1}D_\mu\psi_d\Delta\mu$. Using this result in the second row yields $\Delta\mu=[\alpha I+(D_\nu\psi_d)^{-1}D_\mu\psi_d]^{-1}D_u d^T\Delta u$. Substitute this result in the first row and multiply the row by Δu^T gives

$$\Delta u^T\left\{\overline{H}_{uu}+D_u d[\alpha I+(D_\nu\psi_d)^{-1}D_\mu\psi_d]^{-1}D_u d^T\right\}\Delta u=0. \tag{44}$$

We have $[\alpha I+(D_\nu\psi_d)^{-1}D_\mu\psi_d]^{-1}=\left[\mathrm{diag}(\alpha+\frac{\partial\psi_{d_i}}{\partial\mu_i}(\frac{\partial\psi_{d_i}}{\partial\nu_i})^{-1})\right]^{-1}$, where $\alpha+\frac{\partial\psi_{d_i}}{\partial\mu_i}(\frac{\partial\psi_{d_i}}{\partial\nu_i})^{-1}>0$, $i=1,\ \dots\ ,n_d$. Denote this diagonal matrix as $A=\left[\mathrm{diag}\left(\alpha+\frac{\partial\psi_{d_i}}{\partial\mu_i}\left(\frac{\partial\psi_{d_i}}{\partial\nu_i}\right)^{-1}\right)\right]^{-1}$ with all positive diagonal entries and the matrix product can be represented as $D_u dAD_u d^T$, which is a positive semidefinite matrix. Since $\overline{H}_{uu}$ is also positive definite, Equation (44) yields $\Delta u=0$. It then gives the result that $\Delta u=\Delta\mu=\Delta\nu=\Delta\xi=\Delta\sigma=0$, which contradicts the assumption that $\partial g/\partial z$ is singular.

Therefore, this proof provides sufficient conditions for the BVP-DAEs to be index-1. ■

3 | MULTIPLE SHOOTING SOLUTION

In this article, the BVP-DAEs (37)–(39) are solved by using a multiple shooting method. This is similar to the method developed by Fabien.[6]

First, the overall time interval $[t_\mathrm{i},t_\mathrm{f}]$ is discretized into a mesh with N time nodes such that $t_\mathrm{i}=t_1<t_2<\cdots<t_N=t_\mathrm{f}$. Note that the mesh does not need to be uniform.

At each node t_j, $j=1,2,\ \dots\ ,N$, let $s_j^y\in\mathbb{R}^{n_y}$ represent the differential variables, and $s_j^z\in\mathbb{R}^{n_z}$ represent the algebraic variables. Let p denote the parameter variables of the system. Starting with the initial conditions (s_j^y,s_j^z,p) at time t_j, let $y(t;s_j^y,s_j^z,p)$ denotes the solution to the DAEs (37)–(38) at time t in the interval of interest $[t_j,t_{j+1}]$. Then we require the following conditions be satisfied:

1. At each time node t_j, the following algebraic equations must be satisfied.

$$g(s_j^y,s_j^z,p,\alpha)=0,\quad j=1,2,\ \dots\ ,N. \tag{45}$$

2. At each time node t_j except the last one, the following continuity conditions must be satisfied.

$$y(t_{j+1}; s_j^y, s_j^z, p) - s_{j+1}^y = 0, \quad j = 1, 2, \dots, N-1. \tag{46}$$

3. The following boundary conditions must be satisfied.

$$r(s_1^y, s_N^y, p) = 0. \tag{47}$$

These conditions lead to the system

$$F(s, p, \alpha) = \begin{bmatrix} g(s_1^y, s_1^z, p, \alpha) \\ y(t_2; s_1^y, s_1^z, p) - s_2^y \\ g(s_2^y, s_2^z, p, \alpha) \\ y(t_3; s_2^y, s_2^z, p) - s_3^y \\ \vdots \\ y(t_N; s_{N-1}^y, s_{N-1}^z, p) - s_N^y \\ g(s_N^y, s_N^z, p, \alpha) \\ r(s_1^y, s_N^y, p) \end{bmatrix} = 0, \tag{48}$$

where $s = [s_1^{y\,T}, s_1^{z\,T}, s_2^{y\,T}, s_2^{z\,T}, \dots, s_N^{y\,T}, s_N^{z\,T}]^T \in \mathbb{R}^{n_s}$, $F(s, p, \alpha) \in \mathbb{R}^{n_s+n_p}$, and $n_s = N(n_y + n_z)$.

Thus, an approximate solution to the OCP can be obtained by solving the residual equation $F(s, p, \alpha) = 0$.

3.1 | Solution of the residual equation

A damped Newton's method is used to solve the residual equation. For each $\alpha > 0$, given an initial estimate (s^k, p^k) of the solution to the residual Equation (48), an improved solution (s^{k+1}, p^{k+1}) is obtained from the damped Newton's increment

$$\begin{bmatrix} s^{k+1} \\ p^{k+1} \end{bmatrix} = \begin{bmatrix} s^k \\ p^k \end{bmatrix} + \tau^k \begin{bmatrix} \Delta s^k \\ \Delta p^k \end{bmatrix}, \quad k = 0, 1, \dots, \tag{49}$$

$$DF(s^k, p^k, \alpha) \begin{bmatrix} \Delta s^k \\ \Delta p^k \end{bmatrix} = -F(s^k, p^k, \alpha), \tag{50}$$

where the Jacobian is

$$DF(s, p, \alpha) = \begin{bmatrix} A_1 & C_1 & & & & H_1 \\ & A_2 & C_2 & & & H_2 \\ & & \ddots & \ddots & & \vdots \\ & & & A_{N-1} & C_{N-1} & H_{N-1} \\ B_1 & & & & B_N & H_N \end{bmatrix}, \tag{51}$$

with

$$A_j = \begin{bmatrix} \partial g(s_j^y, s_j^z, p, \alpha)/\partial s_j^y & \partial g(s_j^y, s_j^z, p, \alpha)/\partial s_j^z \\ \partial y(t_{j+1})/\partial s_j^y & \partial y(t_{j+1})/\partial s_j^z \end{bmatrix},$$

$$C_j = \begin{bmatrix} 0 & 0 \\ -I & 0 \end{bmatrix},$$

$$H_j = \begin{bmatrix} \partial g(s_j^y, s_j^z, p, \alpha)/\partial p \\ \partial y(t_{j+1})/\partial p \end{bmatrix}, \quad j = 1, 2, \dots, N-1,$$

$$B_1 = \begin{bmatrix} 0 & 0 \\ \partial r(s_1^y, s_N^y, p)/\partial s_1^y & 0 \end{bmatrix},$$

$$B_N = \begin{bmatrix} \partial g(s_N^y, s_N^z, p, \alpha)/\partial s_N^y & \partial g(s_N^y, s_N^z, p, \alpha)/\partial s_N^z \\ \partial r(s_1^y, s_N^y, p)/\partial s_N^y & 0 \end{bmatrix},$$

$$H_N = \begin{bmatrix} \partial g(s_N^y, s_N^z, p, \alpha)/\partial p \\ \partial r(s_1^y, s_N^y, p)/\partial p \end{bmatrix}.$$

After obtaining $(\Delta s^k, \Delta p^k)$, we need to find the damped factor τ^k to guarantee that for each iteration $\|F(s^{k+1}, p^{k+1}, \alpha)\| < \|F(s^k, p^k, \alpha)\|$, where $s^{k+1} = s^k + \tau^k \Delta s^k$ and $p^{k+1} = p^k + \tau^k \Delta p^k$. The damped factor τ^k is obtained using the classic line-search method. Starting at $\tau^k = 1$ and evaluating the $\|F(s^{k+1}, p^{k+1}, \alpha)\|$, if $\|F(s^{k+1}, p^{k+1}, \alpha)\| < \|F(s^k, p^k, \alpha)\|$ we stop the line-search immediately, or we obtain a new τ^k by dividing it by two $\tau^k_{new} = \tau^k_{old}/2$ and reevaluating the residual term. The maximum number of line-search time is limited. If a damped term τ^k is failed to be found after the maximum number of line-searches, a remesh of the problem is performed as described below.

3.2 | Integration of the DAEs and sensitivity equations

This subsection describes the techniques used to compute the residual and the coefficients of the Jacobian (51). The construction of the residual equation (48) requires the computation of $y(t_{j+1}; s_j^y, s_j^z, p), j = 1, 2, \dots, N-1$.

This paper uses a single step linearly implicit Runge–Kutta (Rosenbrock–Wanner, ROW) method (Hairer and Wanner[9]) to solve the index-1 DAEs. (More details of the implementation of the integrator can be found in Fabien.[6]) The integration is performed as follows.

Define

$$\bar{x}(t) = \begin{bmatrix} y(t) \\ z(t) \end{bmatrix}, \quad \bar{f}(x, p, \alpha) = \begin{bmatrix} h(y(t), z(t), p) \\ g(y(t), z(t), p, \alpha) \end{bmatrix},$$

$$\overline{M} = \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix} \in \mathbb{R}^{(n_y+n_z)\times(n_y+n_z)}, \quad I \in \mathbb{R}^{n_y \times n_y}.$$

Then the index-1 DAEs (37)–(38) is written in the form of $\overline{M}\dot{\bar{x}} = \bar{f}(\bar{x}, p, \alpha)$.

Let $\bar{x}_j = \bar{x}(t_j) = [s_j^{y^T}, s_j^{z^T}]^T = s_j$ be the initial condition for the DAEs at time t_j. The accurate integration of the DAEs requires consistent initial conditions, which should be $g(y(t_j), z(t_j), p, \alpha) = 0$ at the initial time t_j. However, this condition is not guaranteed during the Newton's iteration process.

To ensure that the initial conditions are consistent, a relaxation term is introduced. For example in the interval $[t_j, t_{j+1}]$ the DAEs (37)–(38) is substituted by

$$\overline{M}\dot{\bar{x}} = \bar{f}(\bar{x}, p, \alpha) - \bar{v}_j, \tag{52}$$

where $\bar{v}_j = [0^T, g(s_j^y, s_j^z, p, \alpha)^T]^T \in \mathbb{R}^{n_y+n_z}$ is the relaxation vector. Hence, the initial conditions (s_j^y, s_j^z, p) are consistent for the DAE Equation (52). One major advantage of using this technique is that as the iterations of the Newton's method converge to a solution of the boundary value problem, the relaxation term $\bar{v}_j \to 0$ and the relaxed DAEs becomes the original DAEs.

During integration the Jacobian of $\bar{f}(\bar{x}, p, \alpha)$ with respect to $\bar{x}$ is needed. This Jacobian is defined as

$$\begin{aligned} \overline{W}(\bar{x}, p, \alpha) &= \partial \bar{f}(\bar{x}, p, \alpha)/\partial \bar{x} \\ &= \begin{bmatrix} \partial h(y, z, p)/\partial y & \partial h(y, z, p)/\partial z \\ \partial g(y, z, p, \alpha)/\partial y & \partial g(y, z, p, \alpha)/\partial z \end{bmatrix} \in \mathbb{R}^{(n_y+n_z)\times(n_y+n_z)}. \end{aligned} \tag{53}$$

The inputs to the integration of the DAEs are the relaxed DAEs Equation (52), the time interval of interest $[t_j, t_{j+1}]$, the initial condition $\overline{x}_j$, the parameter p, the mass matrix $\overline{M}$, the Jacobian (53) of the DAEs and the relaxation term $\overline{v}_j$.

Then the approximate solution of the DAEs at time t_{j+1} in time interval $[t_j, t_{j+1}]$, $j = 1, 2, \dots, N-1$ is obtained by the ROW integration (details can be referred at algorithm 3.2 of Fabien[6] (pp. 10)). The set of coefficients used in this paper corresponds to the fourth order ROW method developed by Novati[18] and can be found in Fabien[6] appendix A.

The ROW method also has an embedded formula which is used to estimate the local truncation error. Define $\overline{x}(t_{j+1})$ as the true solution to the DAE (52) with initial condition $\overline{x}_j, p$. The estimated error $\overline{\epsilon}_{j+1} \approx \left\| \overline{x}(t_{j+1}) - \overline{x}_{j+1} \right\|$ using the ROW method can be obtained with the DAE integration together directly.

The construction of the Jacobian matrix $DF(s, p, \alpha)$ (51) requires the trajectory sensitivities $\partial y(t_{j+1})/\partial s_j^y$, $\partial y(t_{j+1})/\partial s_j^z$, and $\partial y(t_{j+1})/\partial p$, $j = 1, 2, \dots, N-1$. These trajectory sensitivities are obtained by integrating the variational equations associated with (52) in the interval $[t_j, t_{j+1}]$. The approach used is described as follows.

Let $\overline{X}(t) = [(\partial \overline{x}/\partial s_j^y)^T, (\partial \overline{x}/\partial s_j^z)^T, (\partial \overline{x}/\partial p)^T]^T \in \mathbb{R}^{(n_y+n_z)\times(n_y+n_z+n_p)}$ represent the sensitivities of $\overline{x}$ with respect to s_j^y, s_j^z, and p. Then using equation (52), it can be seen that in each interval $[t_j, t_{j+1}]$, $j = 1, 2, \dots, N-1$, the trajectory sensitivities must satisfy the linear DAEs

$$\overline{M}\dot{\overline{X}} = \overline{W}(\overline{x}, p, \alpha)\overline{X}(t) + \overline{F}_p(\overline{x}, p, \alpha) - \overline{V}_j, \tag{54}$$

where

$$\overline{F}_p(\overline{x}, p, \alpha) = \begin{bmatrix} 0 & \partial h(y, z, p)/\partial p \\ 0 & \partial g(y, z, p, \alpha)/\partial p \end{bmatrix}, \tag{55}$$

$$\overline{V}_j = \begin{bmatrix} 0 & 0 & 0 \\ \partial g(s_j^y, s_j^z, p, \alpha)/\partial y & \partial g(s_j^y, s_j^z, p, \alpha)/\partial z & \partial g(s_j^y, s_j^z, p, \alpha)/\partial p \end{bmatrix}, \tag{56}$$

and $\overline{W}(\overline{x}, p, \alpha)$ is given in (53). The DAE sensitivities have the initial conditions as

$$\overline{X}_j = \overline{X}(t_j) = \begin{bmatrix} I_1 & 0 & 0 \\ 0 & I_2 & 0 \end{bmatrix},$$

where $I_1 \in \mathbb{R}^{n_y \times n_y}$ and $I_2 \in \mathbb{R}^{n_z \times n_z}$ are both the identity matrices. So as indicated in Equation (52) the initial conditions here are also consistent.

The inputs to the integration of the DAE sensitivities are the relaxed linear DAEs equation (54), the time interval $[t_j, t_{j+1}]$, the initial condition $\overline{x}_j$, the parameter p, the matrix $\overline{M}$, the Jacobian $\overline{W}$ (53) of the DAEs and the relaxation term $\overline{V}_j$. The integration of the DAE sensitivities needs to be done together with the DAE (52) integration.

Then the approximate solution of the DAE sensitivities $\overline{X}_{j+1}$ at time t_{j+1} in time interval $[t_j, t_{j+1}]$, $j = 1, 2, \dots, N-1$ is obtained using the same ROW method to integrate the DAE (54) (details can be referred at algorithm 3.3 of Fabien[6] (pp. 11)).

3.3 | Sequential evaluation of the residual equation and Jacobian

From previous subsections, we have established methods for computing the trajectories $\overline{x}_j$, and the trajectory sensitivities $\overline{X}_j$, $j = 2, 3, \dots, N$. We can first present the sequential method for evaluating the residual equation and the Jacobian. For the computation in each interval $[t_j, t_{j+1}]$, $j = 1, 2, 3, \dots, N-1$, we only need the initial condition x_j at time t_j,the parameters p and the relaxations $\overline{v}_j$ and $\overline{V}_j$ to integrate the DAEs and the DAE sensitivities. By construction the integration from the first time interval $[t_1, t_2]$ to the last time interval $[t_{N-1}, t_N]$ can easily be performed sequentially. Therefore, we can obtain the trajectories $\overline{x}_j$, and the trajectory sensitivities $\overline{X}_j$ at every time node t_j, $j = 2, 3, \dots, N$. The algebraic terms of the DAEs $g(s_j^y, s_j^z, p, \alpha)$ in the residual equation and the derivative terms of the DAEs $\partial g(s_j^y, s_j^z, p, \alpha)/\partial s_j^y$, $\partial g(s_j^y, s_j^z, p, \alpha)/\partial s_j^z$, and $\partial g(s_j^y, s_j^z, p, \alpha)/\partial p$ in the Jacobian are easy to compute sequentially.

3.4 | Parallel evaluation of the residual equation and Jacobian

After establishing the sequential method for computing the trajectories $\overline{x}_j$, and the trajectory sensitivities $\overline{X}_j$, $j = 2, 3, \dots, N$, we can now describe the method for the parallel construction of the residual equation $F(s, p, \alpha)$, and the Jacobian $DF(s, p, \alpha)$.

From Section 3.2, it is easy to notice that the integration done at any time interval is independent of the integration done in any other time interval. So, the integration from t_j to t_{j+1} can be performed simultaneously with the integration at any other interval from t_i to t_{i+1} $(i \neq j)$ with no need for communication or sharing data between the intervals of integration. For other terms in the residual equation and Jacobian, they can be computed in parallel directly by calling the functions with the corresponding inputs.

In this article, we develop a parallel shooting method using the CUDA programming model and the targeted Nvidia GPU to employ the power of the streaming multiprocessors (SMs). The method distributes all the work required to integrate the DAEs (52) and the sensitivities (54) among the GPU. CUDA follows the Single Instruction, Multiple Threads (SIMT) parallel programming model. The one approach that can be used to achieve parallel evaluation is to assign the $N-1$ time intervals onto $N-1$ CUDA threads. Normally, the number of available threads depends on the number of SMs and the number of threads per SM which depend on the architecture of the GPU. So the parallel evaluation of the integration elements of the DAEs and the sensitivities can be realized simply using the CUDA parallel programming model where the implementation details are discussed in Section 4.

3.5 | Linear system solution

As discussed in Section 3.1, the application of the Newton's method to the residual equation (50) requires the solution of a sparse bordered almost block diagonal (BABD) linear system of the form

$$\begin{bmatrix} A_1 & C_1 & & & & H_1 \\ & A_2 & C_2 & & & H_2 \\ & & \ddots & \ddots & & \vdots \\ & & & A_{N-1} & C_{N-1} & H_{N-1} \\ B_1 & & & & B_N & H_N \end{bmatrix} \begin{bmatrix} \Delta s_1 \\ \Delta s_2 \\ \vdots \\ \Delta s_{N-1} \\ \Delta s_N \\ \Delta p \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{N-1} \\ b_N \\ b_p \end{bmatrix}, \tag{57}$$

where $A_j,\ C_j \in \mathbb{R}^{n_{\Delta s} \times n_{\Delta s}},\ H_j \in \mathbb{R}^{n_{\Delta s} \times n_p},\ j = 1, 2, \dots, N-1;\ b_j,\ \Delta s_j \in \mathbb{R}^{n_{\Delta s}},\ j = 1, 2, \dots, N;\ \Delta p,\ b_p \in \mathbb{R}^{n_p},\ B_1,\ B_N \in \mathbb{R}^{(n_{\Delta s}+n_p) \times n_{\Delta s}}$, $H_N \in \mathbb{R}^{(n_{\Delta s}+n_p) \times n_p}$ and $n_{\Delta s} = n_y + n_z$. We should know that (57) is just an expansion of the (50).

3.5.1 | Sequential QR factorization

Fabien[6] introduced a very robust and accurate method of solving this BABD system in parallel which involves the reduction of the coefficient matrix into a sparse upper triangular matrix using orthogonal factorization. The implementation used in this paper is the same as the one described in that paper Fabien[6] (pp. 12 and pp. 17) and we listed necessary steps here.

First, equation (57) is rewritten as

$$\begin{bmatrix} C_1 & & & & A_1 & H_1 \\ A_2 & & & & & H_2 \\ & A_3 & C_3 & & & H_3 \\ & & \ddots & \ddots & & \vdots \\ & & & A_{N-1} & C_{N-1} & H_{N-1} \\ & & & B_N & B_1 & H_N \end{bmatrix} \begin{bmatrix} \Delta s_2 \\ \Delta s_3 \\ \Delta s_4 \\ \vdots \\ \Delta s_N \\ \Delta s_1 \\ \Delta p \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{N-1} \\ b_N \\ b_p \end{bmatrix}, \tag{58}$$

The first block column $[C_1\, A_2]^T$ is transformed into upper triangular shape as

$$\begin{bmatrix} R_1 & E_1 & & & & G_1 & J_1 \\ & \tilde{C}_2 & & & & \tilde{G}_2 & \tilde{H}_2 \\ & A_3 & C_3 & & & & H_3 \\ & & \ddots & \ddots & & & \vdots \\ & & & A_{N-1} & C_{N-1} & & H_{N-1} \\ & & & & B_N & B_1 & H_N \end{bmatrix} \begin{bmatrix} \Delta s_2 \\ \Delta s_3 \\ \Delta s_4 \\ \vdots \\ \Delta s_N \\ \Delta s_1 \\ \Delta p \end{bmatrix} = \begin{bmatrix} d_1 \\ \tilde{b}_2 \\ b_3 \\ \vdots \\ b_{N-1} \\ b_N \\ b_p \end{bmatrix},$$

via QR decomposition as

$$Q_1^T \begin{bmatrix} C_1 \\ A_2 \end{bmatrix} = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}, \quad Q_1^T \begin{bmatrix} C_1 & 0 & A_1 & H_1 & b_1 \\ A_2 & C_2 & 0 & H_2 & b_2 \end{bmatrix} = \begin{bmatrix} R_1 & E_1 & G_1 & J_1 & d_1 \\ 0 & \tilde{C}_2 & \tilde{G}_2 & \tilde{H}_2 & \tilde{b}_2 \end{bmatrix}.$$

By applying the same QR factorization to the remaining column blocks, the BABD system becomes

$$\begin{bmatrix} R_1 & E_1 & & & & G_1 & J_1 \\ & R_2 & E_2 & & & G_2 & J_2 \\ & & R_3 & E_3 & & G_3 & J_3 \\ & & & \ddots & \ddots & & \vdots \\ & & & & R_{N-1} & G_{N-1} & J_{N-1} \\ & & & & & R_N & J_N \\ & & & & & & R_p \end{bmatrix} \begin{bmatrix} \Delta s_2 \\ \Delta s_3 \\ \Delta s_4 \\ \vdots \\ \Delta s_N \\ \Delta s_1 \\ \Delta p \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ \tilde{b}_{N-1} \\ b_N \\ b_p \end{bmatrix}.$$

The unknowns $\Delta s_j, j = 1, 2, \dots, N$, and Δp can then be found by solving the system from the last block-row to the first block-row backwardly using the backward substitution.

3.5.2 | Parallel QR factorization

We listed necessary steps from Fabien[6] (pp. 14 and pp. 17) for the parallel QR factorization for further reference in the paper. More details of the algorithm can be found from the original publication.

Split the original BABD system (57) into M smaller BABD partitions. Take the first partition as an example first as

$$\begin{bmatrix} A_1 & C_1 & & & H_1 \\ & A_2 & C_2 & & H_2 \\ & & \ddots & \ddots & \vdots \\ & & & A_{r_1} \;\; C_{r_1} & H_{r_1} \end{bmatrix} \begin{bmatrix} \Delta s_1 \\ \Delta s_2 \\ \vdots \\ \Delta s_{r_1+1} \\ \Delta p \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{r_1} \end{bmatrix}, \tag{59}$$

All the other partitions have the similar structure.

By applying the algorithm 4.2 from the paper (Fabien[6]) to the partition, the following more compact bordered upper triangular system is obtained as

$$\begin{bmatrix} G_1 & R_1 & E_1 & & & J_1 \\ G_2 & & R_2 & E_2 & & J_2 \\ \vdots & & & \ddots & & \vdots \\ \tilde{A}_{r_1} & & & & \tilde{C}_{r_1} & \tilde{H}_{r_1} \end{bmatrix} \begin{bmatrix} \Delta s_1 \\ \Delta s_2 \\ \vdots \\ \Delta s_{r_1+1} \\ \Delta p \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ \tilde{b}_{r_1} \end{bmatrix}, \tag{60}$$

Taking only the boundary row blocks from those M smaller partition system, the following reduced BABD system is obtained as

$$\begin{bmatrix} \tilde{A}_{r_1} & \tilde{C}_{r_1} & & & & \tilde{H}_{r_1} \\ & \tilde{A}_{r_2} & \tilde{C}_{r_2} & & & \tilde{H}_{r_2} \\ & & \ddots & \ddots & & \\ B_1 & & & & B_N & H_N \end{bmatrix} \begin{bmatrix} \Delta s_1 \\ \Delta s_{r_1+1} \\ \vdots \\ \Delta s_{r_M+1} \\ \Delta p \end{bmatrix} = \begin{bmatrix} \tilde{b}_{r_1} \\ \tilde{b}_{r_2} \\ \vdots \\ \tilde{b}_{r_M} \\ b_N \\ b_p \end{bmatrix}. \tag{61}$$

And, this reduced BABD system can be solved by the sequential QR factorization method introduced in 3.5.1. While after that, all the M smaller partitions can be solved in parallel which saves the computation time to a large extent.

Note that in algorithm 4.3 of the paper (Fabien[6]), the partition backward substitution equation should be updated to

$$\Delta s_j = R_{j-1}^{-1}(d_j - G_{j-1}\Delta s_{r_{\text{start}}} - E_{j-1}\Delta s_{j+1} - J_{j-1}\Delta p).$$

3.6 | Mesh refinement

The OCP is solved primarily on a fixed mesh. However, in some cases we may need to remesh the problem. One remesh criteria is based on the local truncation error (LTE) at each node $\bar{\epsilon}_j, j = 2, 3, \dots, N$. After the Newton's iteration converges, we evaluate the LTE of the DAE integration from the ROW method from Section 3.2 to determine if it exceeds the desired tolerance. Another criteria for remesh is when Newton's iteration fails to find a descent direction for the line-search.

The remesh policy is that if $\bar{\epsilon}_j > \epsilon$ where ϵ is the desired numerical tolerance, the mesh interval $[t_j, t_{j+1}]$ is subdivided by adding one time node to the middle of the time interval or if $\bar{\epsilon}_j > 100\epsilon$, the mesh interval is subdivided by adding three uniformly spaced nodes to the time interval. Linear interpolation is used to estimate the solution at the newly created nodes. Also, if $\bar{\epsilon}_j < \frac{\epsilon}{100}$, we evaluate the residual errors of the 4 subsequent nodes $\bar{\epsilon}_{j+i}, i = 1, 2, 3, 4$ together and if $\bar{\epsilon}_{j+i} < \frac{\epsilon}{100}$, $i = 1, 2, 3, 4$, we delete the time nodes t_{j+1} and t_{j+3}. The remesh policy is sequentially applied on each time interval. The implementation limits the number of nodes that can be added, the minimum number of nodes allowed, and the number of mesh refinements allowed.

3.7 | Continuation method

The continuation method used to solved the constrained optimal control problems is given as follows. The algorithm described above is used to solve the residual equation (48) for a sequence of decreasing parameters α. Starting with an initial estimate $(s^{(0)}, p^{(0)})$ and continuation parameter $\alpha^0 > 0$, the algorithm solves (48) to obtain $(\tilde{s}, \tilde{p})$ that satisfies $\|F(\tilde{s}, \tilde{p}, \alpha^0)\| \le \epsilon$, where ϵ is the desired convergence tolerance. The solution $(\tilde{s}, \tilde{p})$ is then used as the initial estimate for the next iteration where α is decreased by a factor β which is usually between 0.6 and 0.9. The algorithm terminates when $\alpha \le \alpha_m$, where α_m is the desired continuation parameter termination value.

4 | IMPLEMENTATION AND EVALUATION

Algorithms for solving the OCP (7)–(12) are implemented using Python and CUDA. The main functionality provided by this implementation is a set of routines that solve BVP-DAE problems of the form (37)–(39). The codes are directly implemented from all the algorithms described above which can be found at web site https://github.com/UW-OCP/Multiple_Shooting_Solver_CUDA, and all the implementations of the codes are standalone.

The solver is composed of two major parts: (i) construction of the residual with DAEs integration and the Jacobian with the DAE sensitivities integration described in Section 3.2; and (ii) the solver for the BABD linear system described in Section 3.5. Both the sequential and parallel versions of these two parts are implemented to compare the efficiency of the algorithms. To have a fair comparison between the sequential and the parallel solvers, all the linear system solvers (LU, QR) are implemented using python directly instead of using the existing sequential linear system solvers which are implemented using CPython. As shown in the tables below, Solver$_1$ denotes the solver implemented with the sequential implementation of the construction of the residual with DAEs integration and the Jacobian with the DAE sensitivities integration, and sequential implementation of the BABD linear system solving; Solver$_2$ denotes the solver with the parallel implementation of the construction of the residual with DAEs integration and the Jacobian with the DAE sensitivities integration, and parallel implementation of the BABD linear system solving.

The result tables use the following notation.

N_{solution} denotes the number of time nodes in the final solution;

T_{all} denotes the overall running time of the whole problem, where the unit is second (denoted as (s));

M denotes the number of partitions used in solving the BABD system in parallel;

N_{residual} denotes the number of evaluating times of the construction of the residual (48) with DAEs integration;

T_{residual} denotes the running time of the construction of the residual (48) with DAEs integration, where the unit is second (denoted as (s));

N_{Jacobian} denotes the number of evaluating times of the construction of the Jacobian (51) with the DAE sensitivities integration;

T_{Jacobian} denotes the running time of the construction of the Jacobian (51) with the DAE sensitivities integration, where the unit is second (denoted as (s));

N_{BABD} denotes the number of evaluating times of the BABD linear system solving (57);

and T_{BABD} denotes the running time of the BABD linear system solving (57), where the unit is second (denoted as (s)).

The solver is evaluated using more than 140 examples which can be found at website https://github.com/UW-OCP/ocp_test_problems. Here we present two examples from this problem set to illustrate the robustness and efficiency of the solver.

Parallelism. The parallel code in algorithms are implemented using Python and CUDA GPU parallel programming model which is supported by a high performance Python compiler Numba (Lam et al.[19]). Numba supports CUDA by directly compiling a restricted subset of Python code into CUDA kernels and device functions following the CUDA execution model. The detailed explanation of the CUDA working principle and the architecture of Nvidia GPU can be found in CUDA C programming guide (Nvidia[20]) and Storti and Yurtoglu.[11] A brief introduction is given here to provide adequate background for understanding the implementation.

CUDA follows the single instruction, multiple threads (SIMT) parallel programming model. For CUDA programming every function which is designed to run on the GPU is called a CUDA kernel function. The CUDA kernel function specifies a set of instructions to be executed by multiple threads at once (often hundreds or thousands). The assigned threads to execute the kernel function form CUDA warps where threads run simultaneously. Those warps make up CUDA blocks which are independent of each other. CUDA blocks are sent to different Streaming Multiprocessors (SMs), and run concurrently. All threads running on the cores of an SM are executing the same instruction—hence single instruction, multiple threads (SIMT). The modern GPU contains thousands of cores which are capable of performing trillions of floating point operations per second (FLOPs), accelerating the rate at which the computations can be completed. During the execution of the CUDA kernel function, every thread can be positioned by the block index and the thread index so that the thread can perform the instructions on the corresponding portion of the data with the unique index.

When implementing the CUDA kernel function, the primary goal is to divide the overall algorithm into parallel tasks that can be assigned CUDA threads. From the introduction of the parallel evaluation of the residual equation and Jacobian in Section 3.4, it is easy to see that the DAE and DAE sensitivity integrations performed in each time interval $[t_j, t_{j+1}]$, $j = 1, 2, \dots, N-1$ can be assigned to each individual CUDA thread where all those DAE and sensitivity integrations from each individual mesh interval are computed concurrently by CUDA threads.

Figure 1 shows how the CUDA threads are first assigned to each time interval for integrating DAEs, and how the approximated solutions are returned. The solid arrows denote initial set up for each thread by passing the corresponding input while the thin arrows show the return of the approximate solutions after the computation of the integration.

Numerical setup. In all the examples evaluated, the numerical tolerance used in section 3.6 is $\epsilon = 1 \times 10^{-6}$, the initial continuation parameter in section 3.7 is $\alpha_0 = 0.1$, the termination continuation parameter is $\alpha_m = 1 \times 10^{-6}$, the scale factor is $\beta = 0.9$.

Computing environment. All the numerical results given below are performed on a computer with the following hardware and software characteristics. The computer is equipped with Intel® Core$^{\mathring{U}}$ i9-9900K CPU @ 3.60GHz × 16 and GeForce RTX 2070 SUPER/PCIe/SSE2 GPU. The operating system is Ubuntu 19.10.

Example 1. This is a self-excited oscillation system described by van der Pol's equation with two state variables, one control variable, and one state variable inequality constraint from Shimizu and Ito.[21]

$$\min_{x(t)\in\mathbb{R}^2, u(t)\in\mathbb{R}} \int_0^{t_f} 0.5[x_1(t)^2 + x_2(t)^2 + u(t)^2]dt,$$

subject to

$$\dot{x_1}(t) = x_2(t),$$
$$\dot{x_2}(t) = -x_1(t) + (1.0 - x_1(t)^2)x_2(t) + u(t).$$

The initial condition is that $\Gamma = [x_1(0) - 1.0, x_2(0)] = 0$ and there is no final time constraint. The state variable inequality constraint is $-(x_2(t) + 0.25) \leq 0$. Also the final time is $t_f = 5$. The initial estimates for the state variables and control variable are all ones and obtained on a uniform mesh with $N = 101$ nodes.

Using those initial estimates, we obtain the solutions for the differential and algebraic variables that are shown in Figures 2 and 3. The convergence tolerance for this problem is $\epsilon = 10^{-6}$.

Table 1 shows the number of calling times of each algorithm and the computational running time of each algorithm. It also shows the speedup factor which is defined as $S = \frac{T_{\text{sequential}}}{T_P}$, where $T_{\text{sequential}}$ is the running time of the sequential solver and T_M is the running time of the parallel solver with M partitions in BABD system solving.

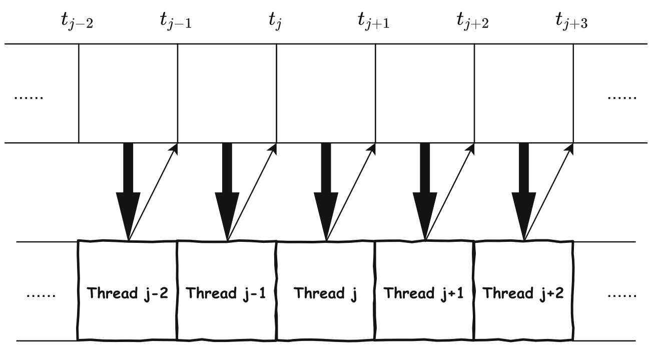

FIGURE 1 Assign CUDA threads to time intervals

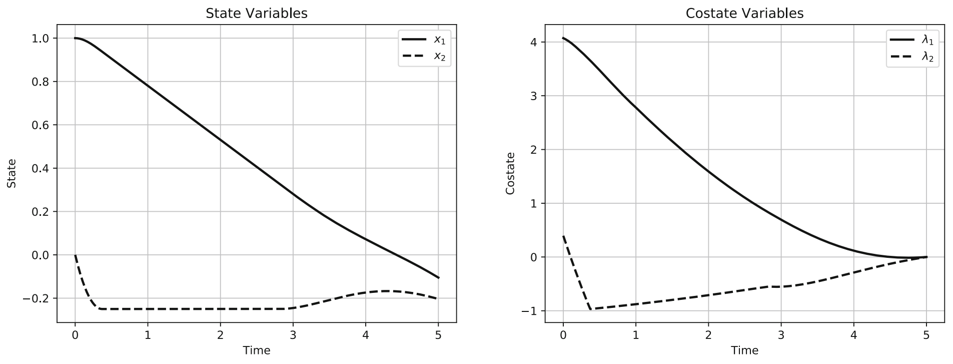

FIGURE 2 Example 1, states and costates

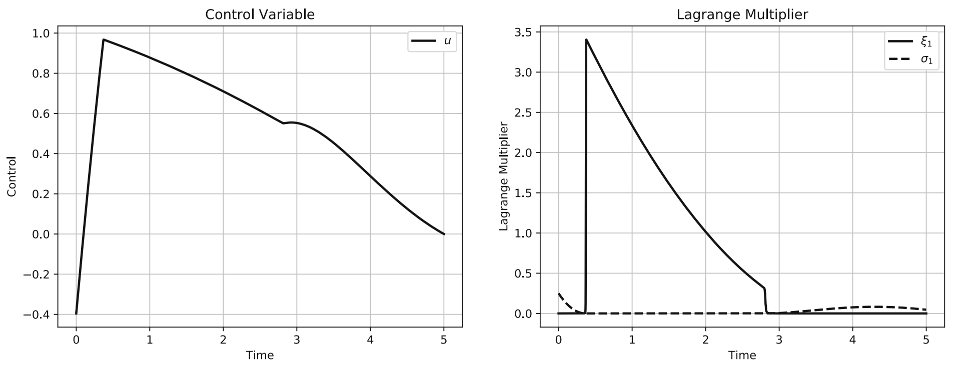

FIGURE 3 Example 1, control and multipliers

TABLE 1 Example 1, evaluating times, running time and speedup factor

	Solver₁	**Solver₂**				
M	**1**	**1**	**4**	**8**	**16**	**32**
N_{residual}	512	512	512	512	512	512
N_{Jacobian}	254	254	254	254	254	254
N_{BABD}	191	191	191	191	191	191
T_{residual}(s)	367.5	9.4	9.7	9.5	9.5	9.5
T_{Jacobian}(s)	1777.6	14.4	14.8	14.7	14.7	14.2
T_{BABD}(s)	403.2	81.7	25.7	16.3	12.9	11.8
N_{solution}	2440	2440	2440	2440	2440	2440
T_{total}(s)	2548.6	105.8	50.5	40.9	37.4	35.8
Speedup		24.1	50.5	62.3	68.1	71.2

The results show that there is a significant boost in speed of the parallel implementation in every part of the algorithm and an overall speedup factor between 24 to 71 are obtained with different number of partitions M used in the BABD system solving. It can be seen that the running time for evaluating residual equation and Jacobian are almost the same for parallel solvers with different partitions M which only affects the BABD linear system solving. As the number of partitions M is increased from 1 to 32, a very significant decrease in the BABD system solving time is realized. It also can be seen that the number of evaluating times for each part of the algorithm and the number of time nodes of the final solution N_{solution} are exactly the same for sequential and parallel solvers. This indicates that although the instructions of the sequential solver happens on CPU which is different from the GPU based parallel solver, their numerical computation is still identical.

Example 2. This problem considers a system with six state variables, four control variables, one parameter variable, and four control variable inequality constraints. We tried to solve the BVP-DAEs using the multiple shooting method both from the OCP without using the penalty function method (see (1)–(6)) on the CVICs, and the OCP using the penalty function method (see (7)–(12)). (Note that the algorithm without the penalty function was unable to converge.) The problem is to minimize the cost functional

$$\min_{u_i\in\mathbb{R},p\in\mathbb{R}} \int_0^1 (p+10)(\rho + u_1(t)^2 + u_2(t)^2 + u_3(t)^2 + u_4(t)^2),$$

subject to

$$\dot{x_1} = (p+10)x_2(t),$$
$$\dot{x_2} = (p+10)((u_1(t)+u_3(t))\cos(x_5(t)) - (u_2(t)+u_4(t))\sin(x_5(t))/m,$$

$$\dot{x_3} = (p+10)x_4(t),$$
$$\dot{x_4} = (p+10)((u_1(t)+u_3(t))\sin(x_5(t)) - (u_2(t)+u_4(t))\cos(x_5(t))/m,$$
$$\dot{x_5} = (p+10)x_6(t),$$
$$\dot{x_6} = (p+10)((u_1(t)+u_3(t))d - (u_2(t)+u_4(t))l)/I,$$

with constants $m = 10.0, d = 5.0, l = 5.0, I = 12.0$, and $\rho = 10.0$. The objective is to find the control inputs $u_i, i = 1, 2, 3, 4$ that move the system from the initial time constraint $\Gamma = [x_1(0), x_2(0), x_3(0), x_4(0), x_5(0), x_6(0)] = 0$ to the final time constraint $\Psi = [x_1(1) - 4.0, x_2(1), x_3(1) - 4.0, x_4(1), x_5(1) - \frac{\pi}{4.0}, x_6(1)] = 0$, while minimizing the cost functional. There are also four inequality constraints that bound the control variables as follows, $-(u_i(t) - 1.5)(-1.5 - u_i(t)) \leq 0, i = 1, 2, 3, 4$.

The initial estimate for this problem is obtained by solving an unconstrained version of the problem and with no bounds on the control inputs. The estimate is obtained on a uniform mesh with $N = 101$ nodes.

Using this initial estimate, we obtain the solutions for the differential and algebraic variables that are shown in Figures 4,5, and 6. The convergence tolerance for this problem is $\epsilon = 10^{-6}$.

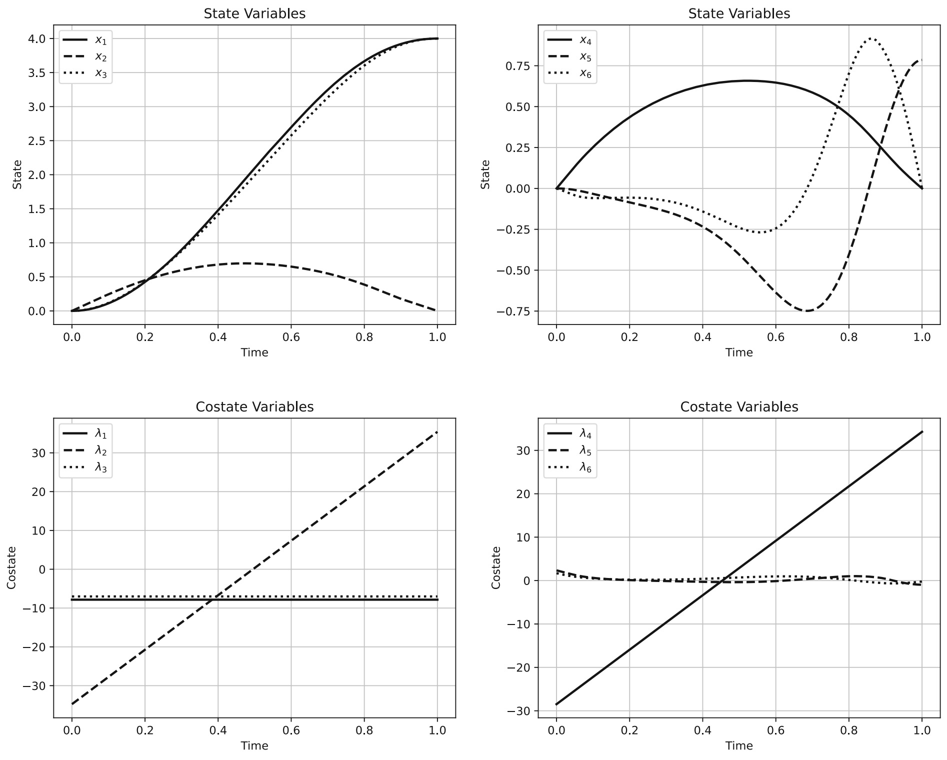

FIGURE 4 Example 2, states and costates

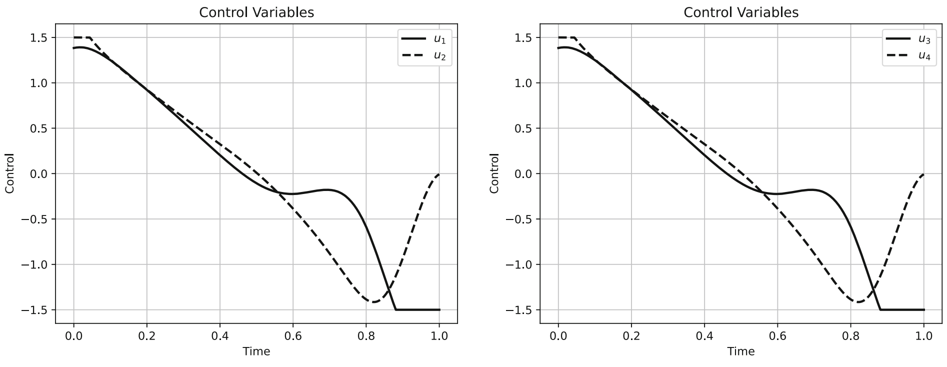

FIGURE 5 Example 2, controls

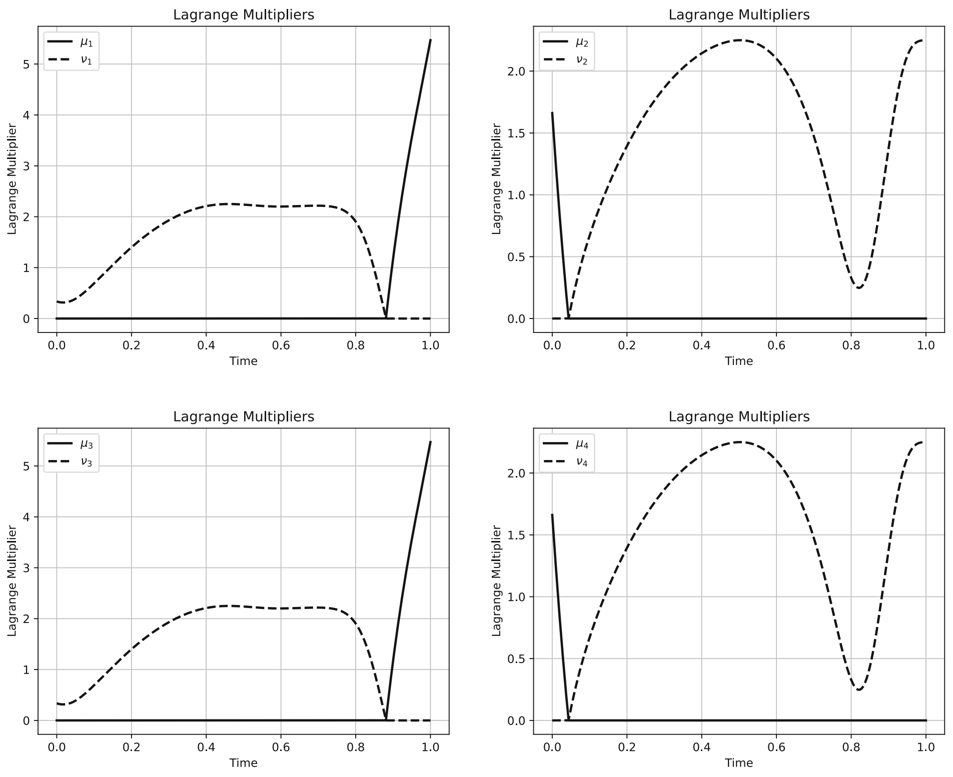

FIGURE 6 Example 2, multipliers

TABLE 2 Example 2, evaluating times, running time and speedup factor

	Solver$_1$	**Solver$_2$**				
M	**1**	**1**	**4**	**8**	**16**	**32**
N_{residual}	627	627	627	627	627	627
N_{Jacobian}	332	332	332	332	332	332
N_{BABD}	221	221	221	221	221	221
T_{residual}(s)	2236.7	17.6	17.6	17.5	17.5	17.6
T_{Jacobian}(s)	42,362.9	115.0	114.7	113.8	114.8	114.6
T_{BABD}(s)	9020.0	2485.6	751.1	559.4	354.6	253.0
N_{solution}	641	641	641	641	641	641
T_{total}(s)	12,772.4	2618.4	883.7	691.0	487.2	385.5
Speedup		20.5	60.7	77.6	110.1	139.1

From Table 2, we can see that both solvers converge to the same solution with a mesh of $N_{\text{solution}} = 641$ nodes and every part of the algorithm has a significant boost in performance with an overall speedup factor from 20 to 139.

The resultant BVP-DAEs contains $n_y = 13$ differential variables, $n_z = 12$ algebraic variables, and $n_p = 13$ parameter variables. For the mesh of 641 nodes, the corresponding Jacobian matrix (51) of the linear system is of size 16,025 by 16,025. Also, to have an apple-to-apple comparison between the sequential and parallel solvers, all the linear system solver (LU, QR) are implemented in python directly instead of calling Numpy libraries which is implemented using CPython. Also, the continuation method in Section 3.7 requires decent amount of iterations to reach the tolerance. Those factors contribute to the long computation time for the sequential solver and also shows the significant performance improvement of the parallel solver. Moreover, as the problem contains several active CVICs, a dense mesh is needed. The BABD system becomes bigger with the dense mesh size and solving that is also very expensive. The speedup for the BABD system solving is also very remarkable which contributes to the high speedup improvement for the GPU parallel implementation compared with the CPU sequential implementation.

5 | CONCLUSIONS

This paper presents an indirect method for solving optimal control problems with control variable inequality constraints, state variable inequality constraints, and parameters. Here a slacked unconstrained penalty function method is used to treat the inequality constraints.

By decreasing the penalty term gradually to zero, it is also shown that the optimum of the cost functional given by the transformed problem converges to the optimum of the cost functional of the original optimal control problem. By the transformation, the BVP-DAEs given by the first-order necessary conditions are guaranteed to be index-1. The BVP-DAEs are solved by a multiple shooting method in which a damped Newton's continuation method is used to solve the residual equation by collecting the local residual on each time node after discretization.

The paper also presents a fast and accurate GPU-based version of the solver. A numerical solver is implemented using Python and CUDA with the use of GPU parallel computing capability. The algorithm implements the part that can be processed independently in parallel and decrease the running time for solving the problem significantly. With the increase of the computing ability of the computer, a further reduced running time is also possible. Comparing the performance of the sequential and parallel implementation of the algorithm, it can be seen there is a significant speedup in the parallel implementation. For some complicated problems with many variables and complicated inequality constraints, the parallel algorithm is very efficient.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are openly available in *ocp_test_problems* at https://github.com/UW-OCP/ocp_test_problems and *Multiple_Shooting_Solver_CUDA* at https://github.com/UW-OCP/Multiple_Shooting_Solver_CUDA.

ORCID

Chaoyi Yang https://orcid.org/0000-0002-7917-3136

REFERENCES

1. Bock HG, Plitt KJ. A multiple shooting algorithm for direct solution of optimal control problems. *IFAC Proc Vol*. 1984;17(2):1603-1608.
2. Fabien BC. Some tools for the direct solution of optimal control problems. *Adv Eng Softw*. 1998;29(1):45-61.
3. Fabien BC. Direct optimization of dynamic systems described by differential-algebraic equations. *Opt Control Appl Methods*. 2008;29(6):445-466.
4. Gerdts M. Global convergence of a nonsmooth Newton method for control-state constrained optimal control problems. *SIAM J Optim*. 2008;19(1):326-350.
5. Fabien BC. Indirect solution of inequality constrained and singular optimal control problems via a simple continuation method. *J Dyn Syst Meas Control*. 2014;136(2):021003.
6. Fabien BC. Parallel indirect solution of optimal control problems. *Opt Control Appl Methods*. 2014;35(2):204-230.
7. Kanzow C. Some noninterior continuation methods for linear complementarity problems. *SIAM J Matrix Anal Appl*. 1996;17(4):851-868.
8. Fabien BC. A noninterior continuation method for constrained optimal control problems. Proceedings of the Control Conference (ECC), 2016 European IEEE; 2016:1598-1603; IEEE.
9. Hairer E, Wanner G. Solving ordinary differential equations II: Stiff and differential-algebraic problems second revised edition with 137 figures. Springer Series in Computational Mathematics 1996;14.
10. Amodio P, Cash J, Roussos G, et al. Almost block diagonal linear systems: sequential and parallel solution techniques, and applications. *Numer Linear Algebra Appl*. 2000;7(5):275-317.
11. Storti D, Yurtoglu M. CUDA for engineers: an introduction to high-performance parallel computing. Addison-Wesley Professional; 2015.
12. Nickolls J, Buck I, Garland M. Scalable parallel programming. Proceedings of the 2008 IEEE Hot Chips 20 Symposium (HCS); 2008:40-53; IEEE.
13. Luebke D. CUDA: scalable parallel programming for high-performance scientific computing. Proceedings of the 2008 5th IEEE International Symposium on Biomedical Imaging: from NANO to Macro; 2008:836-838; IEEE.
14. Fabien BC. Indirect numerical solution of constrained optimal control problems with parameters. Proceedings of the American Control Conference; Vol. 3, 1995:2075-2076; IEEE.
15. Yang C, Fabien BC. An adaptive mesh refinement method for indirectly solving optimal control problems. *Numer Algorithms*. 2022;1-33.

16. Agrawal SK, Fabien BC. *Optimization of Dynamic Systems*. Vol 70. Springer Science & Business Media; 2013.
17. Hartl RF, Sethi SP, Vickson RG. A survey of the maximum principles for optimal control problems with state constraints. *SIAM Rev*. 1995;37(2):181-218.
18. Novati P. Some secant approximations for Rosenbrock W-methods. *Appl Numer Math*. 2008;58(3):195-211.
19. Lam SK, Pitrou A, Seibert S. Numba: a llvm-based python jit compiler. Proceedings of the 2nd Workshop on the LLVM Compiler Infrastructure in HPC ACM; 2015:7.
20. Nvidia. CUDA. C programming guide; 2019; Nvidia Corporation https://docs.nvidia.com/cuda/cuda-c-programming-guide/
21. Shimizu K, Ito S. Constrained optimization in Hilbert space and a generalized dual quasi-Newton algorithm for state-constrained optimal control problems. *IEEE Trans Automat Contr*. 1994;39(5):982-986.

How to cite this article: Yang C, Fabien BC. Parallel solution of optimal control problems using the graphics processing unit. *Optim Control Appl Meth*. 2023;44(1):2-22. doi: 10.1002/oca.2927