



## Article

# GPU-Accelerated PD-IPM for Real-Time Model Predictive Control in Integrated Missile Guidance and Control Systems

Sanghyeon Lee <sup>1</sup>, Heoncheol Lee <sup>2,\*</sup>, Yunyoung Kim <sup>3</sup>, Jaehyun Kim <sup>3</sup> and Wonseok Choi <sup>3</sup>

<sup>1</sup> Research Institute of Manufacturing and Productivity, Kumoh National Institute of Technology, Gumi 39177, Gyeongbuk, Korea; freeg159@kumoh.ac.kr

<sup>2</sup> Department of IT Convergence Engineering, Kumoh National Institute of Technology, Gumi 39177, Gyeongbuk, Korea

<sup>3</sup> Precision Guided Munition R&D Laboratory, LIGNEX1, Seongnam 13488, Gyeonggi, Korea; yunyoung.kim@lignex1.com (Y.K.); jaehyun.kim@lignex1.com (J.K.); wonseok.choi@lignex1.com (W.C.)

\* Correspondence: hlee@kumoh.ac.kr; Tel.: +82-54-478-7458

**Abstract:** This paper addresses the problem of real-time model predictive control (MPC) in the integrated guidance and control (IGC) of missile systems. When the primal-dual interior point method (PD-IPM), which is a convex optimization method, is used as an optimization solution for the MPC, the real-time performance of PD-IPM degenerates due to the elevated computation time in checking the Karush–Kuhn–Tucker (KKT) conditions in PD-IPM. This paper proposes a graphics processing unit (GPU)-based method to parallelize and accelerate PD-IPM for real-time MPC. The real-time performance of the proposed method was tested and analyzed on a widely-used embedded system. The comparison results with the conventional PD-IPM and other methods showed that the proposed method improved the real-time performance by reducing the computation time significantly.

**Keywords:** graphics processing unit; primal-dual interior point method; model predictive control; real-time systems; integrated missile guidance and control



**Citation:** Lee, S.; Lee, H.; Kim, Y.; Kim, J.; Choi, W. GPU-Accelerated PD-IPM for Real-Time Model Predictive Control in Integrated Missile Guidance and Control Systems. *Sensors* **2022**, *22*, 4512. <https://doi.org/10.3390/s22124512>

Academic Editor: Andrey V. Savkin

Received: 22 April 2022

Accepted: 9 June 2022

Published: 14 June 2022

**Publisher's Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Copyright:** © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

When evaluating the performance of a missile system, its guidance and control system is the main factor to be considered. Traditional guidance laws, such as proportional navigation guidance law, generate acceleration commands under the given target-missile kinematics. The commands are followed by an autopilot system that generates actuator commands to achieve the desired acceleration. In general, guidance and control are designed separately without considering interactions between the two systems. Although a separated design principle has proven to be reliable and effective over the decades, the method shows degradation in combined response compared to the separated conditions. This is because traditional guidance laws cannot guarantee optimal characteristics under autopilot lags and dynamic constraints. In [1], for example, the circular navigation guidance law that theoretically promises zero miss-distance was proposed. Although their method showed a robust performance even under uncertain autopilot models, its performance has been inevitably constrained by autopilot and dynamic response.

Integrated guidance and control (IGC) design concept has been considered as an alternative to solve the afore-mentioned issues. An IGC design is a single system that performs the role of both guidance and control, generating fin commands based on missile and target states. Since control commands are designed considering the interaction between guidance and control loop, IGC has the potential to enhance missile performance. Additionally, it is helpful to reduce the number of iterations and costs for the entire design process. To perform IGC, various control techniques are introduced. From classical control techniques to various nonlinear control techniques, including feedback linearization [2],

sliding mode control [3,4], backstepping control [5], dynamic surface technique [6–8], and optimization-based methods [9] have been applied to solve the problem. Shima et al. [3] proposed a sliding mode control (SMC)-based IGC, enhancing the robustness of overall systems. Shtessel and Tournes [4] extended the research to a higher-order SMC to attenuate the chattering problem for dual control missiles. In [6], Hou and Duan utilized dynamic surface control (DSC) for the IGC problem under unmatched uncertainties. The DSC-based IGC technique is expanded by Liang et al. [7], with the additional consideration of input saturation. Kim et al. [9] proposed an explicit solution of finite time-varying state feedback control with a feedforward term.

Of many optimization-based methods, the Model Predictive Control (MPC) technique has been thought to be a powerful solution for IGC [10–15]. Compared to the other control techniques utilized in the previous research [2–9], online MPC provides the optimal solution within certain state constraints. MPC produces a control input that minimizes the objective function specified on the receding prediction horizon. The technique repeatedly solves the finite horizon open-loop optimal control problem and implements it in the form of closed-loop control. It can be applied not only to linear time-invariant systems, but also to multivariable, time-varying nonlinear systems [16]. In general, the optimal control problem is a quadratic programming (QP) problem. As it is an explicit solution to the time-varying state feedback form, it is easy to be adopted on-board. Additionally, MPC has the advantage of being able to set state and output constraints. Especially for the missile terminal guidance phase, the acceleration limit and seeker field-of-view (FOV) are crucial constraints caused by the finite maneuver capacity and seeker's image plane. It is essential to consider these limits as inequality constraints in the optimization problem. Considering the advantages, MPC is a suitable control technique for terminal homing guidance.

Despite its outstanding performance, applications of online MPC have been limited to slow dynamic systems because of computational bottlenecks. The issue is mainly caused by the optimization process that requires excessive computational capacities. To ease the problem, various studies on optimization algorithms and acceleration methods are conducted. In particular, convex optimization algorithms have been considered as a conducive solution for their computational efficiency and parallelizable characteristic. Gradient-based convex optimization techniques, such as the alternating direction method of multipliers (ADMM) [17–19], primal-dual interior point method (PD-IPM), parallel quadratic programming (PQP) [20,21], and active set method (ASM) [22–24], are employed. In this work, PD-IPM [24,25], which is the most commonly used technique for convex optimization, is applied. PD-IPM is developed using the Newton direction of the optimality conditions for the logarithmic barrier problem. The method simultaneously updates primal and dual variables by setting a residual function. Compared to ASM and PQP, PD-IPM requires a smaller number of iterations to reach the desired convergence level [26,27]. Additionally, the PD-IPM technique satisfies strict interior point feasibility by adopting a backtracking line search. This eases the constraint that the initial point must be feasible.

However, despite the high efficiency of PD-IPM, MPC for the IGC problem needs further improvements for real-time implementation. As the dynamics of the missile and target show fast responses, the update rate-of-control command should be large enough for stability and to yield a smaller miss distance [28]. Furthermore, the large size of the prediction horizon is required for precise interception performance. Consequentially, the optimization process in MPC for IGC demands frequent operations of multiplication and inversion for large-sized matrix. For this reason, we adopt the parallel design for real-time GPU implementation. Research on accelerating the PD-IPM is conducted, as shown in Table 1. Even though there is limited research [29–36] that deals with the real-time problem of PD-IPM, it focuses on the acceleration of the linear equation solver part of PD-IPM. However, except for the linear equation solver part, we found that the KKT condition construction part also requires considerable computation time. Moreover, there is no related work that applies the PD-IPM to IGC systems.

**Table 1.** Works related to the acceleration of PD-IPM.

| Related works | Target Device | MPC | Parallelization Part   | IGC Application |
|---------------|---------------|-----|------------------------|-----------------|
| [29]          | GPU           | ○   | Linear equation solver | ×               |
| [30–34]       | GPU           | ×   | Linear equation solver | ×               |
| [35]          | GPU           | ×   | None                   | ×               |
| [36]          | FPGA          | ○   | Linear equation solver | ×               |

In this paper, we propose a GPU-accelerated PD-IPM method, which is conducted in MPC for real-time IGC systems, which parallelizes the KKT condition construction part to reduce the computation time of the PD-IPM. A series of complex matrix operations are performed on the KKT condition construction. The proposed method transforms these complex matrix operations into easier forms in the context of parallelization. Then, the transformed matrices are reformed to sparse matrices. Finally, parallelization is conducted with the sparse matrices through both built-in and customized CUDA kernels. The contributions of this paper are as follows.

- This is the first approach to accelerate missile MPC on GPU.
- The problem of considerable computation time in the KKT condition construction part of PD-IPM is firstly addressed and analyzed.
- A new parallelization method is developed for the KKT condition construction part of PD-IPM.
- The computation time for PD-IPM is significantly reduced, even considering the overhead time for the CUDA (Compute Unified Device Architecture) initialization on a widely-used embedded system.

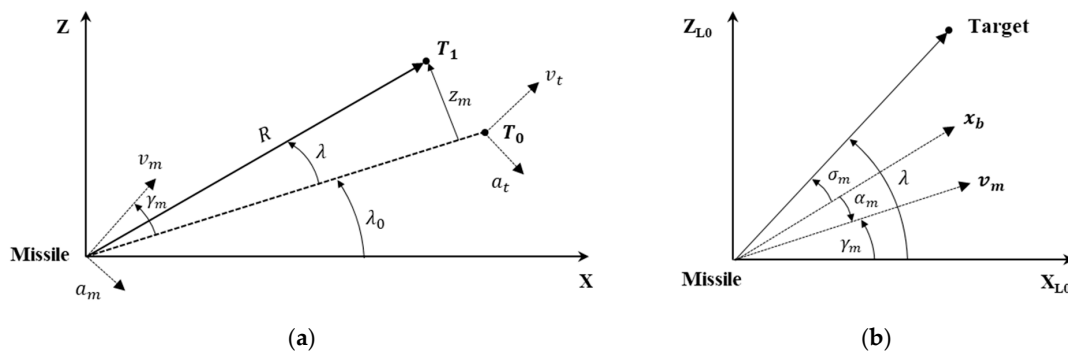
The remainder of this paper is organized as follows. Section 2 describes the optimization problem of the IGC system and MPC with PD-IPM to solve it. Additionally, the real-time problem of PD-IPM is addressed. In Section 3, after computation times for PD-IPM are profiled in a block-wise manner, a new parallelization method for the KKT condition construction part of PD-IPM is proposed. In Section 4, the evaluation results of the proposed method are shown and quantitatively compared with other methods on a widely-used embedded system. Finally, Section 5 presents the conclusions.

## 2. Problem Description

For the IGC problem, we considered missile terminal homing phase geometry in a two-dimensional plane. Figure 1a depicts planar homing engagement geometry, where the subscripts  $m$  and  $t$  denote the missile and target. Reference coordinate system  $X-Z$  is centered at the missile's center of gravity; initial target position  $T_0$  and deviated target position  $T_1$  are defined on the reference coordinate. Initial line-of-sight (LOS) angle  $\lambda_0$ , LOS angle displacement from initial LOS frame  $\lambda$ , and range-to-go  $R$  are also represented. Missile acceleration, velocity, and flight-path angle are denoted by  $a_m$ ,  $v_m$ ,  $\gamma$ , respectively. Relative displacement  $z_m$  is defined as a normal distance between the target position and initial LOS. In Figure 1b, the seeker look angle  $\sigma_m$ , angle of attack  $\alpha_m$ , and body-fixed coordinate system  $x_b$  are denoted. Reference coordinate frame  $X_{L0}-Z_{L0}$  is the initial LOS frame whose origin is also located at the missile's center of gravity. It is assumed that, in the terminal homing phase, the distance between the missile and target is small enough so that linearization can be performed on the initial LOS frame. Additionally, missile velocity is assumed to be constant.

The main objective of terminal homing is actuating the missile to intercept the target under the finite maneuver capacity and seeker look-angle limit. In addition, based on the previous study [13], the look-angle rate is limited in bound to prevent image distortion and signal intensity reduction problems. With the acceleration limit  $a_{max}$ , look-angle limit  $\sigma_{max}$ , and look-angle rate limit  $\dot{\sigma}_{max}$ , the constraints can be expressed as follows:

$$-a_{max} \leq a_m \leq a_{max}, -\sigma_{max} \leq \sigma_m \leq \sigma_{max}, -\dot{\sigma}_{max} \leq \dot{\sigma}_m \leq \dot{\sigma}_{max} \quad (1)$$



**Figure 1.** Missile terminal homing phase geometry in a two-dimensional plane (a) Planar engagement geometry; (b) engagement geometry defined on initial LOS frame.

### 2.1. Augmented Model for Integrated Guidance and Control

Considering the missile short-period dynamics, kinematics, and actuator dynamics, augmented continuous equations for IGC are given by [9,12,13]

$$\underbrace{\begin{bmatrix} \dot{\delta}_m \\ \dot{\alpha}_m \\ \dot{q}_m \\ \dot{\gamma}_m \\ \dot{z}_m \end{bmatrix}}_{\dot{x}} = \underbrace{\begin{bmatrix} -\omega_a & 0 & 0 & 0 & 0 \\ Z_\delta & Z_\alpha & 1 & 0 & 0 \\ M_\delta & M_\alpha & M_q & 0 & 0 \\ -Z_\delta & -Z_\alpha & 0 & 0 & 0 \\ 0 & 0 & 0 & v_m & 0 \end{bmatrix}}_{\bar{A}} \underbrace{\begin{bmatrix} \delta_m \\ \alpha_m \\ q_m \\ \gamma_m \\ z_m \end{bmatrix}}_x + \underbrace{\begin{bmatrix} \omega_a \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{\bar{B}} \underbrace{\delta_c}_u \quad (2)$$

where  $q_m$  is pitch rate,  $\delta_c$  is control fin command, and  $\delta_m$  is actuator response.  $Z_\delta$ ,  $Z_\alpha$ ,  $M_\delta$ ,  $M_\alpha$ ,  $M_q$ ,  $Z_\delta$  and  $Z_\alpha$  are aerodynamic dimensional derivatives. The actuator dynamics are modeled as a 1st-order lag system with time constant  $1/\omega_a$ .

Equation (2) is characterized by its input and state variables. Compared to conventional guidance and control design, augmented equations for IGC simultaneously consider target-missile kinematics and dynamics. For simplicity, state variable vector and input are represented as  $x$ ,  $u$ . System and input matrices are denoted as  $\bar{A} \in \mathbf{R}^{5 \times 5}$ ,  $\bar{B} \in \mathbf{R}^{5 \times 1}$ . Equation (2) is discretized with sampling interval  $\Delta t$ .

$$x_{k+1} = Ax_k + Bu_k \quad (3)$$

System and input matrices of the discretized equation are  $A = e^{\Delta t \bar{A}}$ ,  $B = \left( \int_0^{\Delta t} e^{\tau \bar{A}} d\tau \right) \bar{B}$ , respectively. The notation  $k$  represents sampling time step.

As mentioned above, control input  $u$  should be generated within the extent that it does not violate the restrictions. Inequality constraints defined in Equation (1) are linearized and expressed in matrix form [12,13]. As shown below, linearized matrix  $C_k$  is time-varying.  $R_k$  is range-to-go in  $k$ th time step.

$$\underbrace{\begin{bmatrix} v_m Z_\delta & v_m Z_\alpha & 0 & 0 & 0 \\ -v_m Z_\delta & -v_m Z_\alpha & 0 & 0 & 0 \\ 0 & -1 & 0 & -1 & 1/R_k \\ 0 & 1 & 0 & 1 & -1/R_k \\ 0 & 0 & -1 & -v_m/R_k & -(v_m + v_t)/R_k^2 \\ 0 & 0 & 0 & v_m/R_k & (v_m + v_t)/R_k^2 \end{bmatrix}}_{C_k} \underbrace{\begin{bmatrix} \delta_m \\ \alpha_m \\ q_m \\ \gamma_m \\ z_m \end{bmatrix}}_x \leq \underbrace{\begin{bmatrix} a_{max} \\ a_{max} \\ \sigma_{max} \\ \sigma_{max} \\ \dot{\sigma}_{max} \\ \dot{\sigma}_{max} \end{bmatrix}}_d \quad (4)$$

$$R_k = R_0 - k(v_m + v_t)\Delta t \quad (5)$$



**Algorithm 1.** *Cont.*


---

```

 $\theta = \min\left(1, \min\left(-u_k^i / \Delta u_k^i : \Delta u_k^i < 0\right)\right)$ 
// Backtracking Line Search to find  $\theta$ 
while  $\|r(y^+, u^+, v^+)\| > (1 - \alpha\theta)\|r(y, u, v)\|$ 
   $y^+ = y^k + \theta\Delta y^k, u^+ = u^k + \theta\Delta u^k, v^+ = v^k + \theta\Delta v^k$ 
  compute  $f^+ = \tilde{C}y^+ - \tilde{D}, r_{dual}^+, r_{prim}^+, r_{cent}^+$ 
   $\theta = \alpha\theta$ 
// Primal-Dual Update
 $(y^{k+1}, u^{k+1}, v^{k+1}) := (y^k + \theta\Delta y^k, u^k + \theta\Delta u^k, v^k + \theta\Delta v^k)$ 

```

---

**3. Proposed Method****3.1. Overview of the Proposed Method**

As shown in Figure 2, the PD-IPM algorithm applied to this application is divided into four parts: (1) calculate residues, (2) construct modified KKT matrix, (3) calculate search direction, and (4) backtracking line search. To improve the computation speed of the algorithm, we measured the computation time for each part of PD-IPM and proposed a method of partially accelerating the parts that required improvements.

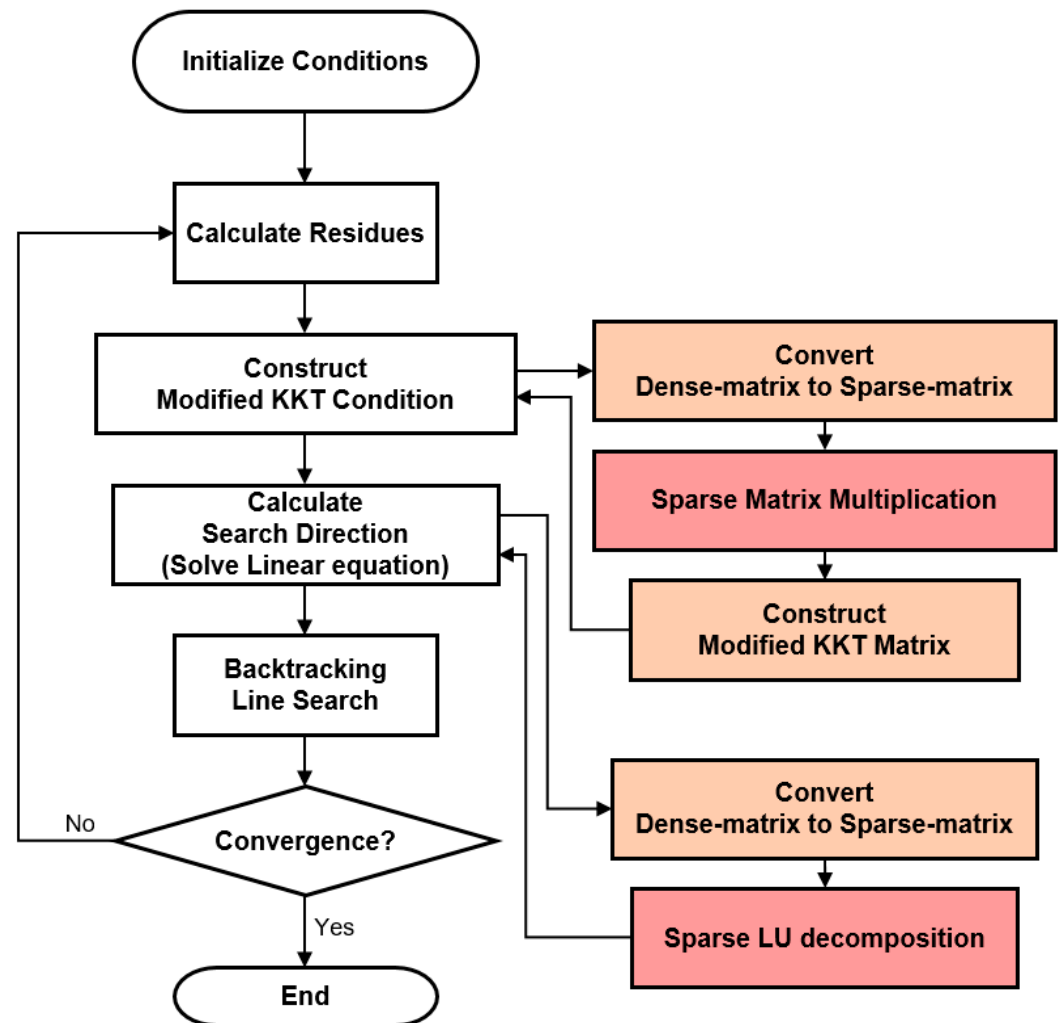


Figure 2. The flowchart of the proposed method.

### 3.2. Computation-Time Profiling

Computation-time measurements for each part were performed in Nvidia Jetson Xavier NX (20W 6CORE mode), and the results are summarized in Table 2, showing that the construct modified KKT condition and calculate search direction parts take more time than the other two parts. In addition, detailed computation times for these two parts were measured, and the results are shown in Tables 3 and 4, respectively.

**Table 2.** The computation time for each part of PD-IPM.

| Part                             | Computation Time (ms) |
|----------------------------------|-----------------------|
| Calculate residue                | 287.285               |
| Construct modified KKT condition | 1202.956              |
| Calculate search direction       | 1612.328              |
| Backtracking line search         | 352.767               |

**Table 3.** The computation time for each process of construct modified KKT condition.

| Part                                  | Computation Time (ms) |
|---------------------------------------|-----------------------|
| Convert dense matrix to sparse matrix | 78.513                |
| Sparse matrix multiplication          | 695.494               |
| Construct modified KKT matrix         | 321.575               |
| etc.                                  | 108.922               |

**Table 4.** The computation time for each process of calculate search direction.

| Part                                  | Computation Time (ms) |
|---------------------------------------|-----------------------|
| Convert dense matrix to sparse matrix | 187.086               |
| Solve linear equation                 | 1425.109              |

First, Table 3 shows that the sparse matrix multiplication process and construct modified KKT matrix process take a lot of time in the construct modified KKT condition part. Therefore, we set these two parts as parallelization sections and accelerated them to improve the performance. Next, Table 4 shows that the solve linear equation process takes a lot of time in the calculate search direction part. This process is implemented with the SparseLU class in the Eigen library to solve a linear equation using Sparse LU (Lower–Upper) Decomposition. We simply replaced this process with the CUDA cusolver library.

### 3.3. Parallelization Based on CSR and CSC

The modified KKT matrix is obtained through the following matrix operation:

$$S = \frac{1}{2} \left( \begin{bmatrix} 2P - A^T B & C^T \\ C & 0 \end{bmatrix} + \begin{bmatrix} 2P - A^T B & C^T \\ C & 0 \end{bmatrix}^T \right) \quad (9)$$

where  $S$  is the modified KKT matrix,  $P$  is the covariance matrix,  $A$  is the inequality constraint,  $C$  is the equality constraint matrix, and  $B$  is the matrix calculated from the penalty function and equality constraint matrix. To make it easier to compute in parallel, Equation (9) can be simplified as follows.

$$S = \begin{bmatrix} P + P^T - \frac{1}{2}(A^T B) - \frac{1}{2}(A^T B)^T & C^T \\ C & 0 \end{bmatrix} \quad (10)$$

Then, we divide the top-left sub-matrix of matrix  $S$  into Equations (11) and (12).

$$D = -\frac{1}{2}(A^T B) \quad (11)$$

$$M = P + P^T + D + D^T \tag{12}$$

Matrices  $A$  and  $B$  are large matrices of sizes of about  $600 \times 600$ . The multiplication of such a large matrix takes a long time to complete. However, if the matrix contains many zero elements, it can be converted to a sparse matrix to reduce unnecessary operations and the computation time. The Compressed Sparse Row (CSR) and Compressed Sparse Column (CSC) are commonly used formats for sparse matrices, and the conversion examples are shown in Figure 3, respectively. For the CSR format, the accumulated number of non-zero data per row is stored in the ptr array, the column index of non-zero data is stored in the index array, and the element is stored in the data array. As a result, three one-dimensional arrays are created. The CSC format is converted similarly to the CSR format, except that the row changes into a column. As the conversion example shows, the CSR format uses row-wise indexing, whereas the CSC format uses column-wise indexing. Additionally, in matrix multiplication, since the left and right matrices are accessed row-wise and column-wise, respectively, we applied CSR and CSC formats to the left and right matrices, respectively, to increase the matrix access speed. In addition, for a faster operation, sparse matrices  $A$  and  $B$  are sorted, and matrix  $A$  is transposed in advance for the convenience of calculation.

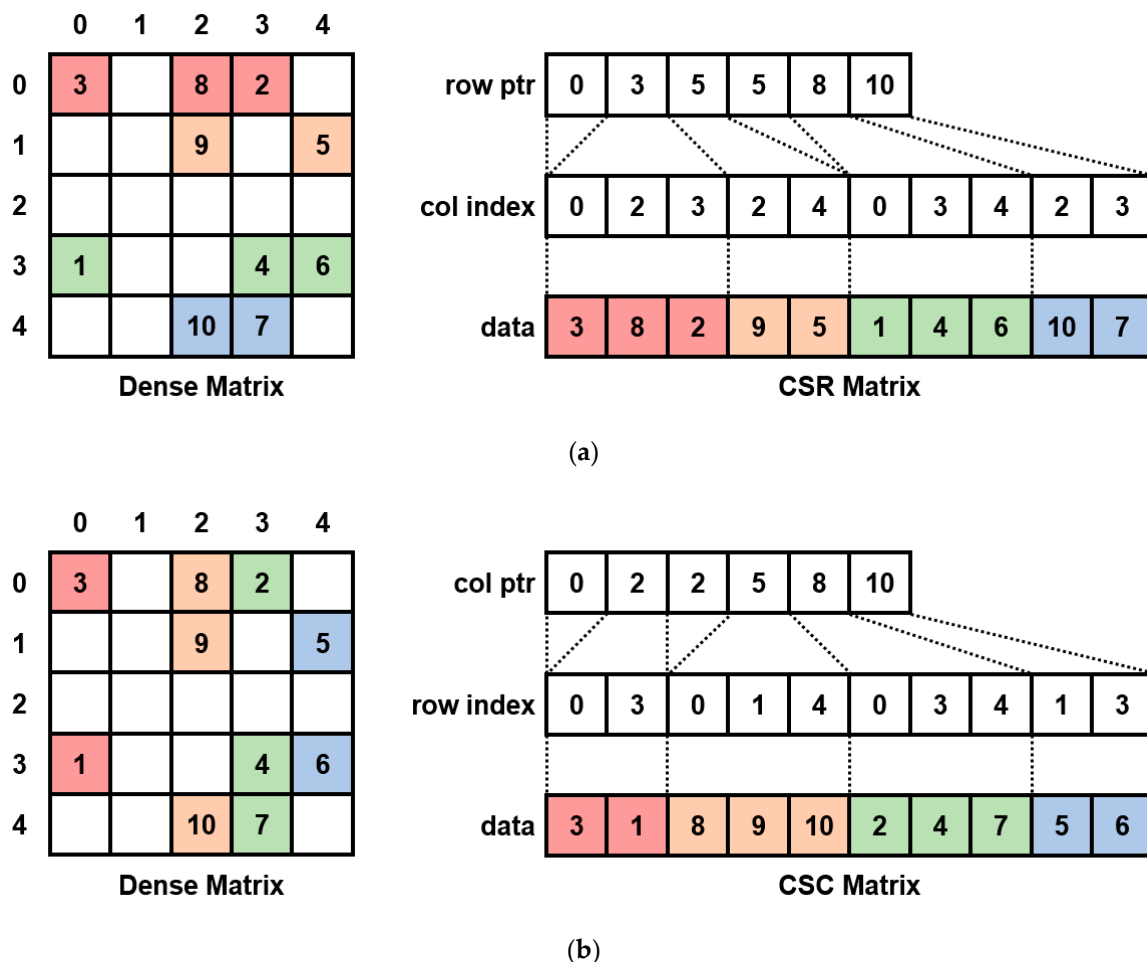


Figure 3. The examples of sparse-matrix conversions: (a) Dense to CSR conversion; (b) dense to CSC conversion.

The parallelization algorithm of Equation (11) is shown in Algorithm 2.  $A_d$  is the non-zero element of matrix  $A$ ,  $A_i$  is the column index for non-zero elements of matrix  $A$ , and  $A_p$  is the cumulative number of non-zero elements for each row of matrix  $A$ .  $B_d$  is a non-zero element of matrix  $B$ ,  $B_i$  is the row index for non-zero elements of matrix  $B$ , and  $B_p$



is the cumulative number of non-zero elements for each column of matrix  $B$ . Additionally, the result matrix  $D$  is stored as a dense matrix.

The parallel sparse matrix multiplication algorithm shown in Algorithm 2 works as follows. First, a two-dimensional thread is created equal to the size of matrix  $D$ , and the algorithm is executed in parallel (Line 1). The non-zero elements of row  $r$  of matrix  $A$  are compared with the non-zero elements of column  $c$  of matrix  $B$ , and multiplication is performed when the column index of matrix  $A$  is equal to the row index of matrix  $B$  (Lines 7–8). If the row index of matrix  $B$  is greater than the column index of matrix  $A$ , it means that there is no element with the same index because the matrix is sorted, so the loop is terminated (Lines 5–6). After all the loops are finished, the  $sum$  variable, in which the multiplication of the  $(r, c)$  element is stored, is calculated to satisfy Equation (11), and finally stored in  $D$  (Line 12).

The parallelization algorithm of Equation (12) is shown in Algorithm 3, and the matrix  $M$  is calculated using the covariance matrices  $P$  and  $D$ , which is calculated in the parallel sparse matrix multiplication algorithm. The parallel construct modified KKT matrix (top-left sub-matrix) algorithm shown in Algorithm 3 works by adding elements of matrices  $P$  and  $D$ , and their transpose matrices in parallel (Line 2).

---

#### Algorithm 2. Parallel Sparse-Matrix Multiplication

---

|        |                                                                                                                                                                                              |
|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Input  | * CSR format matrix A(data: $A_d$ , col index: $A_i$ , row ptr: $A_p$ )<br>* CSC format matrix B(data: $B_d$ , row index: $B_i$ , col ptr: $B_p$ )<br>* Sparse matrices A, B must be sorted. |
| Output | Dense matrix $D(D)$                                                                                                                                                                          |
| 1.     | all $D(r, c)$ do, in parallel:                                                                                                                                                               |
| 2.     | $sum \leftarrow 0$                                                                                                                                                                           |
| 3.     | for $i \leftarrow A_p(r)$ to $A_p(r+1)$ do:                                                                                                                                                  |
| 4.     | for $j \leftarrow B_p(c)$ to $B_p(c+1)$ do:                                                                                                                                                  |
| 5.     | if $A_i(i) < B_i(j)$ then:                                                                                                                                                                   |
| 6.     | break                                                                                                                                                                                        |
| 7.     | else if $A_i(i) = B_i(j)$ then:                                                                                                                                                              |
| 8.     | $sum \leftarrow sum + A_d(i) * B_d(j)$                                                                                                                                                       |
| 9.     | end if                                                                                                                                                                                       |
| 10.    | end for                                                                                                                                                                                      |
| 11.    | end for                                                                                                                                                                                      |
| 12.    | $D(r, c) \leftarrow -sum/2$                                                                                                                                                                  |
| 13.    | end                                                                                                                                                                                          |

---

#### Algorithm 3. Parallel Construct Modified KKT Matrix (top-left sub-matrix)

---

|        |                                                            |
|--------|------------------------------------------------------------|
| Input  | Dense matrix $P(P)$ , dense matrix $D(D)$                  |
| Output | dense matrix $M(M)$                                        |
| 1.     | all $M(r, c)$ do in parallel:                              |
| 2.     | $M(r, c) \leftarrow P(r, c) + P(c, r) + D(r, c) + D(c, r)$ |
| 3.     | end                                                        |

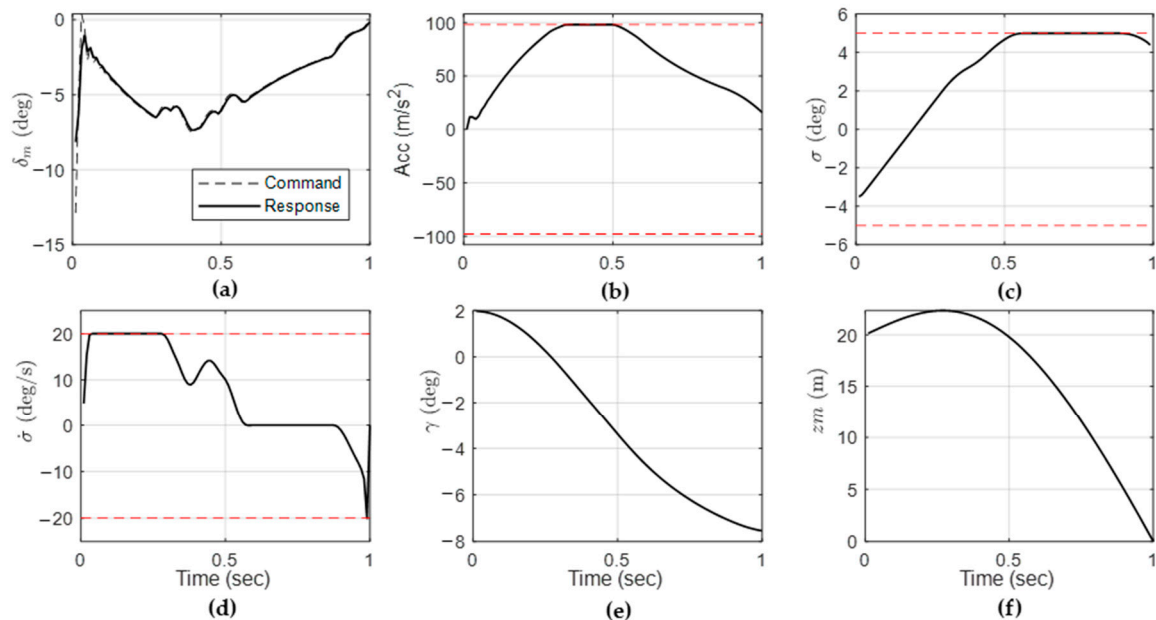
---

## 4. Results

### 4.1. Simulation Results

Numerical simulation was performed to compare the computation time. For comparison, terminal homing engagement was assumed. The size of the finite horizon was 1 s, with a sampling time of 0.01 s. Aerodynamic coefficients of missile were set as  $Z_\delta = -0.2105$ ,  $Z_\alpha = -3.1316/s$ ,  $M_\delta = 160/s^2$ ,  $M_\alpha = -234/s^2$ ,  $M_q = -5/s$ . The constant velocities of missile and target were  $v_m = 380$  m/s and  $v_T = 380$  m/s. Fin actuator response was modeled with  $\omega_a = 100/s$ . The initial-state variable of the missile was set to  $\gamma_{m0} = 2$  deg,  $z_{m0} = 20$  m. Inequality constraint parameters were  $a_{max} = 10$  G,  $\sigma_{max} = 5$  deg, and  $\dot{\sigma}_{max} = 20$  deg/s.

Figure 4 shows the single-step simulation results obtained under given design parameters. The optimization problem in Equations (7) and (8) were solved using PD-IPM. The red dotted line on the graph represents the given constraints of acceleration, seeker look angle, and look-angle rate. It is shown that the target was successfully intercepted within the given limitations.



**Figure 4.** (a) actuator command and response; (b) acceleration of missile; (c) seeker look angle; (d) look-angle rate; (e) flight path angle; and (f) relative displacement between the target and the missile.

#### 4.2. Results of Algorithm Acceleration with GPU

The test was conducted in Nvidia Jetson Xavier NX (20W 6CORE mode), and parallelization was implemented through CUDA 10.2. We compared the following four implementations: (1) CPU only; (2) CUDA dense: The matrix multiplication section was implemented and parallelized as dense-matrix multiplication; (3) CUDA SpMM: the matrix multiplication section was implemented and parallelized as sparse-matrix multiplication using the csrmm (CSR  $\times$  CSR) function in CUDA cusparse library, and (4) ours.

Figure 5, Table 5 show the computation time comparison results for the construct modified KKT condition part, and Figure 6, Table 6 show the computation time comparison results for the entire application. In the solve linear equation part, the CPU only was applied with the Eigen library, and the other three methods were improved using the cusolverSpcsrslvluHost function of the CUDA cusolver library. The CUDA initialization delay is a delay that occurs when calling the CUDA API and initializing the GPU. Therefore, it occurs only once during the entire application runtime and is not directly related to the algorithm.

#### 4.3. Analysis

First, Figure 5 and Table 5 show that the CUDA dense and CPU only have almost the same performance. This indicates that parallelizing dense-matrix multiplication makes no sense, since the matrices are very sparse. The CUDA SpMM using the unsorted CSR  $\times$  CSR multiplication showed about twice the performance compared to the CPU only. However, due to the use of a heavy library, cusparse, there was a long delay problem of the CUDA initialization. On the other hand, our proposed method, Ours, uses CSR  $\times$  CSC multiplication to improve row and column access speed. Additionally, the CUDA initialization delay is less than the CUDA SpMM because it does not use any additional libraries. As a

result, the performance was about 4 times faster than the CPU only, and even including the CUDA initialization delay, the performance was about 3 times faster.

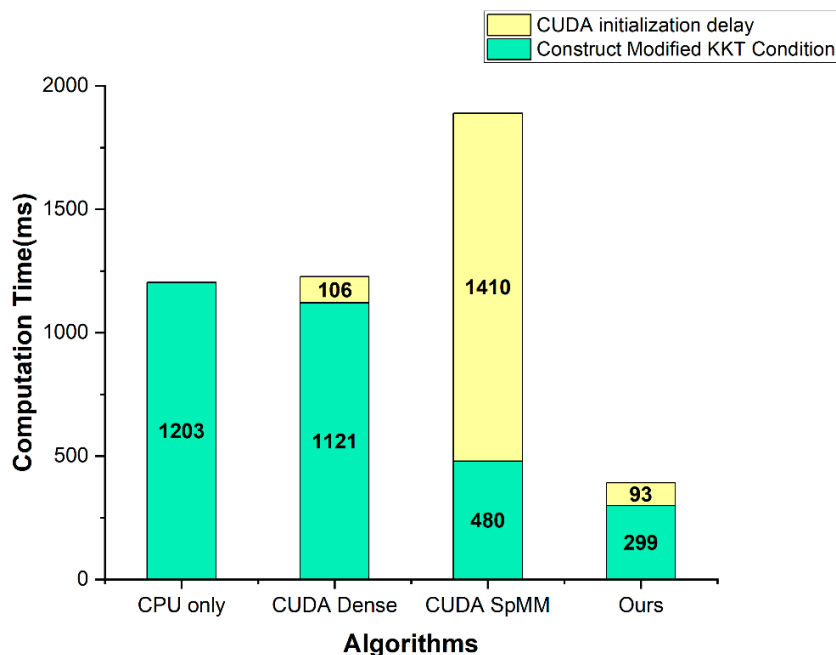


Figure 5. The computation time comparison result graph of the construct modified KKT condition part.

Table 5. The computation time comparison results of the construct modified KKT condition part.

| Part                             | Computation Time(ms) |            |           |         |
|----------------------------------|----------------------|------------|-----------|---------|
|                                  | CPU Only             | CUDA Dense | CUDA SpMM | Ours    |
| CUDA initialization delay        | -                    | 106.152    | 1409.594  | 92.593  |
| Construct modified KKT condition | 1202.956             | 1120.698   | 479.706   | 298.689 |
| Total                            | 1202.956             | 1226.85    | 1889.3    | 391.282 |

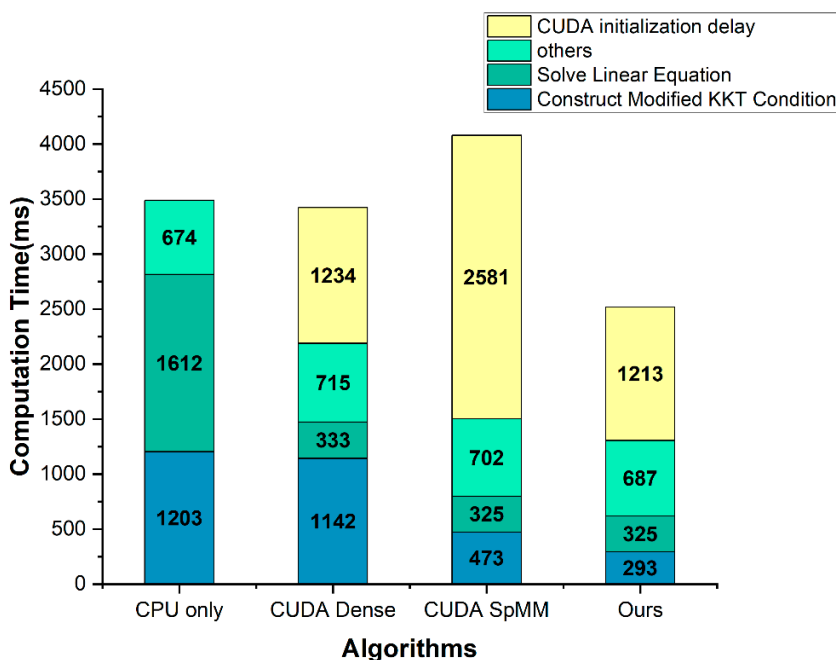


Figure 6. The computation time comparison result graph of the entire application.

**Table 6.** The computation time comparison results of the entire application.

| Part                              | Computation Time (ms) |            |           |          |
|-----------------------------------|-----------------------|------------|-----------|----------|
|                                   | CPU only              | CUDA Dense | CUDA SpMM | Ours     |
| CUDA initialization delay         | -                     | 1234.11    | 2580.534  | 1213.339 |
| Others                            | 673.505               | 714.613    | 701.644   | 687.447  |
| Solve linear equation             | 1612.328              | 332.588    | 325.038   | 325.44   |
| Construct modified KKT condition  | 1202.956              | 1141.699   | 473.369   | 293.119  |
| Total                             | 3488.789              | 3423.01    | 4080.585  | 2519.345 |
| without CUDA initialization delay | 3488.789              | 2188.9     | 1500.051  | 1306.006 |

Next, Figure 6 and Table 6 show that the performance of the solve linear equation part in the CUDA dense, CUDA SpMM, and Ours was improved by using the CUDA cusolver library function. However, since the cusolver library is heavy, the CUDA initialization delay increased accordingly. Compared to the CUDA only, the proposed method performed about 2.7 times faster and about 1.4 times faster when the CUDA initialization delay was included.

#### 4.4. Discussion

In this paper, we focused on the parallelization of construct modified KKT condition part in a PD-IPM solver. As shown in Table 2, the construct modified KKT condition part is not the most time-consuming part of the entire computation process. However, it is the part where the efficiency of the parallel operation can be maximized because most of the operations that the construct modified KKT condition part contains are fixed-size sparse-matrix operations.

According to the first result, as shown in Figure 5, our method is more efficient for embedded systems. In general, the CUDA initialization delay does not occupy a large part of the computation time in the desktop environment. However, in an embedded device with a relatively low performance, this delay may take more time than the computation time of the algorithm. In fact, the CUDA SpMM method presented this problem. Our method is suitable not only for desktops, but also for embedded devices because of the short delay. The second result, as shown in Figure 6, indicates that the need for the acceleration of the construct modified KKT condition part as well as the solve linear equation part, and demonstrates that our method works effectively. Because the solve linear equation part takes the most time, other studies have concentrated on that part and improved its performance. However, in an extensive matrix system, such as the IGC, the construct modified KKT condition part also takes a lot of computation time. Therefore, we focused on the acceleration of the construct modified KKT condition part and attained a significant performance improvement.

By adopting CSC and CSR parallelization methods, the computation time of the entire optimization process was significantly reduced. Considering that the CUDA initialization delay occurred just one time, the reduced amount was about 62.5% for the whole flight. This improvement in the context of computation time is highly promising to apply the real-time missile control system. Additionally, since the CUDA used in the proposed method is a widely-used library for algorithm acceleration, the accomplishment of this paper can be applied to other research areas, such as robotics [37–39].

## 5. Conclusions

This paper dealt with the problem of real-time model predictive control (MPC) in integrated guidance and control (IGC) of missile systems. The problem of much computation time in the KKT condition construction part of PD-IPM was firstly addressed and analyzed. A new GPU-based parallelization method was proposed for the KKT condition construction part of PD-IPM. The computation time for PD-IPM was significantly reduced, even considering the overhead time for the CUDA initialization on a widely-used embedded system.

The comparison results with the conventional PD-IPM and other methods showed that the proposed method improved the real-time performance by reducing the computation time significantly. In future studies, algorithm acceleration using other hardware, such as FPGA, will be conducted. Additionally, the proposed method will be extended to more algorithms with closed-loop performances, and the stability of a given MPC approach will be evaluated.

**Author Contributions:** All authors contributed the present paper with the same effort in finding available literatures and writing the paper. S.L., H.L. designed and implemented the proposed method. Y.K., J.K. and W.C. described the system and problems. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was supported by Theatre Defense Research Center funded by Defense Acquisition Program Administration under Grant UD200043CD.

**Institutional Review Board Statement:** Not available.

**Informed Consent Statement:** Not available.

**Data Availability Statement:** Not available.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Manchester, I.R.; Savkin, A.V. Circular navigation missile guidance with incomplete information and uncertain autopilot model. *J. Guid. Control. Dyn.* **2004**, *27*, 1078–1083. [[CrossRef](#)]
2. Menon, P.K.; Ohlmeyer, E.J. Integrated design of agile missile guidance and autopilot systems. *Control. Eng. Pract.* **2001**, *9*, 1095–1106. [[CrossRef](#)]
3. Shima, T.; Idan, M.; Golan, O.M. Sliding-mode control for integrated missile autopilot guidance. *J. Guid. Control. Dyn.* **2006**, *29*, 250–260. [[CrossRef](#)]
4. Shtessel, Y.B.; Tournes, C.H. Integrated higher-order sliding mode guidance and autopilot for dual control missiles. *J. Guid. Control. Dyn.* **2009**, *32*, 79–94. [[CrossRef](#)]
5. He, S.; Song, T.; Lin, D. Impact Angle Constrained Integrated Guidance and Control for Maneuvering Target Interception. *J. Guid. Control. Dyn.* **2017**, *40*, 2652–2660. [[CrossRef](#)]
6. Hou, M.; Duan, G. Adaptive block dynamic surface control for integrated missile guidance and autopilot. *Chin. J. Aeron.* **2013**, *26*, 741–750. [[CrossRef](#)]
7. Liang, X.; Hou, M.; Duan, G. Adaptive dynamic surface control for integrated missile guidance and autopilot in the presence of input saturation. *J. Aerosp. Eng.* **2015**, *28*, 04014121. [[CrossRef](#)]
8. Liu, W.; Wei, Y.; Duan, G. Barrier Lyapunov Function-based Integrated Guidance and Control with Input Saturation and State Constraints. *Aerosp. Sci. Technol.* **2019**, *84*, 845–855. [[CrossRef](#)]
9. Kim, J.H.; Whang, I.H.; Kim, B.M. Finite horizon integrated guidance and control for terminal homing in vertical plane. *J. Guid. Control. Dyn.* **2016**, *39*, 1103–1111. [[CrossRef](#)]
10. Bachtiar, V.; Manzie, C.; Kerrigan, E.C. Nonlinear model predictive integrated missile control and its multi-objective tuning. *J. Guid. Control. Dyn.* **2017**, *40*, 2958–2967. [[CrossRef](#)]
11. Chai, R.; Savvaris, A.; Chai, S. Integrated Missile Guidance and Control Using Optimization-based Predictive Control. *Nonlinear Dyn.* **2019**, *96*, 997–1015. [[CrossRef](#)]
12. Park, J.H.; Kim, Y.I.; Kim, J.H. Integrated Guidance and Control Using Model Predictive Control with Flight Path Angle Prediction against Pull-Up Maneuvering Target. *Sensors* **2020**, *20*, 3143. [[CrossRef](#)] [[PubMed](#)]
13. Kim, T.H.; Park, J.H.; Kim, J.H. Computational Issues in Sparse and Dense Formulations of Integrated Guidance and Control with Constraints. *Int. J. Aeronaut Space Sci.* **2020**, *21*, 826–835. [[CrossRef](#)]
14. Ma, L.; Shan, J.; Liu, J.; Ding, Y. Missile IGC Based on Improved Model Predictive Control and Sliding Mode Observer. *Int. J. Aerosp. Eng.* **2021**, 2021. [[CrossRef](#)]
15. Shamaghdari, S.; Nikraves, S.K.Y.; Haeri, M. Integrated guidance and control of elastic flight vehicle based on robust MPC. *Int. J. Robust Nonlinear Control* **2015**, *25*, 2608–2630. [[CrossRef](#)]
16. Mohsen, H.; Amin, R.; Wenjun, Z. An interpolation-based model predictive controller for input–output linear parameter varying systems. *Int. J. Dyn. Control.* **2022**, 1–14. [[CrossRef](#)]
17. Gabay, D.; Mercier, B. A dual algorithm for the solution of nonlinear variational problems in finite-element approximations. *Comput. Math. Appl.* **1976**, *2*, 17–40. [[CrossRef](#)]
18. East, S.; Cannon, M. ADMM for MPC with state and input constraints, and input nonlinearity. In Proceedings of the 2018 annual American control conference (ACC), Milwaukee, WI, USA, 27–29 June 2018.

19. Danielson, C. An alternating direction method of multipliers algorithm for symmetric MPC. *IFAC-PapersOnLine* **2018**, *51*, 319–324. [[CrossRef](#)]
20. Brand, M.; Shilpiekandula, V.; Yao, C.; Bortoff, S.A. A Parallel Quadratic Programming Algorithm for Model Predictive Control. *IFAC Proc.* **2011**, *44*, 1031–1039. [[CrossRef](#)]
21. Yu, L.; Goldsmith, A.M.; Di Cairano, S. Efficient Convex Optimization on GPUs for Embedded Model Predictive Control. In Proceedings of the 10th General Purpose GPUs, Austin, TX, USA, 4–8 February 2017.
22. Glad, T.; Jonson, H. A method for state and control constrained linear quadratic control problems. *IFAC Proc.* **1984**, *17*, 1583–1587. [[CrossRef](#)]
23. Richter, S.; Jones, C.N.; Morari, M. Real-time input-constrained MPC using fast gradient methods. In Proceedings of the 48th IEEE Conference on Decision and Control (CDC) Held Jointly with 2009 28th Chinese Control Conference, Shanghai, China, 16–18 December 2009.
24. Boyd, S.; Vandenberghe, L. *Convex Optimization*; Cambridge University Press: Cambridge, UK, 2004; pp. 324–615.
25. Lustig, I.J.; Marsten, R.E.; Shanno, D.F. Computational experience with a primal-dual interior point method for linear programming. *Linear Algebra Its Appl.* **1991**, *152*, 191–222. [[CrossRef](#)]
26. Lau, M.S.; Yue, S.P.; Ling, K.V.; Maciejowski, J.M. A comparison of interior point and active set methods for FPGA implementation of model predictive control. In Proceedings of the 2009 European Control Conference (ECC), Budapest, Hungary, 23–29 August 2009.
27. Abughalieh, K.M.; Alawneh, S.G. A survey of parallel implementations for model predictive control. *IEEE Access* **2019**, *7*, 34348–34360. [[CrossRef](#)]
28. Zarchan, P. *Tactical and Strategic Missile Guidance*, 6th ed.; American Institute of Aeronautics and Astronautics: Reston, VA, USA, 2012; pp. 473–527.
29. Nicolai, F.; Gade, N. Interior Point Methods on GPU with Application to Model Predictive Control. Available online: [https://backend.orbit.dtu.dk/ws/portalfiles/portal/103047513/phd338\\_Gade\\_Nielsen\\_NF.pdf](https://backend.orbit.dtu.dk/ws/portalfiles/portal/103047513/phd338_Gade_Nielsen_NF.pdf). (accessed on 8 June 2022).
30. Smith, E.; Gondzio, J.; Hall, J. GPU Acceleration of the Matrix-Free Interior Point Method. *Lect. Notes Comput. Sci.* **2012**, *7203*, 681–689.
31. Jin, H.J.; O’leary, D. Implementing an interior point method for linear programs on a CPU-GPU system. *Electron. Trans. Numer. Anal.* **2008**, *28*, 174–189.
32. Jing, J.; Xianggao, C.; Xiaola, L. Efficient SVM Training Using Parallel Primal-Dual Interior Point Method on GPU. In Proceedings of the 2013 International Conference on Parallel and Distributed Computing, Applications and Technologies, London, UK, 3–5 July 2013.
33. Maggioni, M. Sparse Convex Optimization on GPUs. Ph.D. Thesis, University of Illinois at Chicago, Chicago, IL, USA, 2016. Available online: <https://hdl.handle.net/10027/20173> (accessed on 8 June 2022).
34. Shah, U.A.; Yousaf, S.; Ahmad, M.O. On the Efficiency of Supernodal Factorization in Interior-Point Method Using CPU-GPU Collaboration. *IEEE Access* **2020**, *8*, 120892–120904. [[CrossRef](#)]
35. Legendre, M.; Moussaoui, S.; Idier, J.; Schmidt, F. Parallel implementation of a primal-dual interior-point optimization method for fast abundance maps estimation. In Proceedings of the 2013 5th Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS), Gainesville, FL, USA, 26–28 June 2013.
36. Liu, J.; Peyrl, H.; Burg, A.; George, A. FPGA implementation of an interior point method for high-speed model predictive control. In Proceedings of the 24th International Conference of Field Programmable Logic and Applications (FPL), Munich, Germany, 2–4 September 2014.
37. Chen, Z.; Li, J.; Wang, S.; Wang, J.; Ma, L. Flexible gait transition for six wheel-legged robot with unstructured ter-rains. *Robot. Auton. Syst.* **2022**, *150*, 1–18.
38. Pipatpaibul, P.; Ouyang, P.R. Application of Online Iterative Learning Tracking Control for Quadrotor UAVs. *Int. Sch. Res. Not.* **2013**, *2013*, 476153. [[CrossRef](#)]
39. Wang, F.; Qian, Z.; Yan, Z.; Yuan, C.; Zhanga, W. Novel Resilient Robot: Kinematic Analysis and Experimentation. *IEEE Access* **2019**, *8*, 2885–2892. [[CrossRef](#)]