A Hardware Implementable Two-Level Parallel Computing Algorithm for General Minimum-Time Control

Shin-Yeu Lin

Abstract—**In this paper, we propose a hardware implementable two-level parallel computing algorithm for general minimum-time control. We first discretize and transform the minimum-time control problem for a continuous-time system into a parameter optimization problem which is large dimensional and nonseparable. Then, the proposed two-level algorithm decomposes this parameter optimization problem into a master-slave problem. The master problem can be easily solved by a one-dimensional gradient method, and the slave problem will be solved by the proposed parallel computing method which combines recursive quadratic programming with the dual method. Furthermore, we have proved the convergence of this iterative two-level parallel computing algorithm under some conditions. Based on the VLSI array processor technology, we present a dedicated hardware computing architecture to realize this algorithm. The corresponding time complexity is also analyzed. Finally, several practical problems including the minimum-time orbit transfer problem and the minimum-time robot control problem have been simulated. The results show that the algorithm is well-suited for real-time application of minimum-time control.**

I. Introduction and Problem Statement

MINIMUM-time control is an important class of optimal control problems. For such problems, numerous sequential computing techniques have been developed [1]–[5]. In general, these techniques take considerable computation time for complicated constrained minimum-time control problems, and special care is needed for the systems with discontinuities, for example, the *bang-bang problem* [3, p. 190], [6].

The purpose of this paper is to present a *hardware implementable two-level parallel computing algorithm* for general minimum-time control. The idea of this algorithm is novel. The problem under consideration can be any complicated nonlinear, multivariable constrained minimum-time control problem. The realizability of this algorithm by *VLSI array processors* has great appeal to *real-time processing* systems.

Mathematically, the general minimum-time control problem can be expressed as

$$\min_{u(t),0\le t\le t_f} t_f$$

subject to

$$\dot{x}(t) = f'(x(t), u(t), t),\ x(0) = x_0$$
$$T(x(t_f), t_f) = 0,\ g(x(t), u(t)) \le 0 \qquad (1)$$

where $u \in \mathbb{R}^p$ are control variables; $x \in \mathbb{R}^n$ are state variables; f': $\mathbb{R}^{n+p+1} \to \mathbb{R}^n$ is the vector function of the state equations; T: $\mathbb{R}^{n+1} \to \mathbb{R}^m$, $m \le n$ is a vector function, and the surface $T(x(t_f), t_f) = 0$ denotes the target set; g: $\mathbb{R}^{n+p} \to \mathbb{R}^q$ is the vector function of the inequality constraints.

The problem is to find a p-component control $u(t)$, $0 \le t \le t_f$ to transfer the system from the initial state x_o at $t = 0$ to the target set $T(x(t_f), t_f) = 0$ in minimum time t_f, while satisfying the constraints.

In our approach, we first discretize the above minimum-time control problem, then transform the resultant discretization problem by adding *slack variables* s_i, $i = 0, \cdots, N$ to the equality constraints and *penalizing* those slack variables in the objective function with a large positive *penalty coefficient* M. The final parameter optimization problem as shown in (2) will well approximate the original minimum-time control problem if the time interval $\Delta t = t_f/N$ in the discretization is small enough and the penalty coefficient M in the transformation is sufficiently large [7].

$$\min t_f + M\sum_{i=0}^{N} s_i^T s_i$$
$$x_{i+1} - x_i - \frac{t_f}{N} f(x_i, u_i, i, t_f) + s_i = 0, \qquad x_0 = x_o$$
$$T(x_N, t_f) + s_N = 0,\ g(x_i, u_i) \le 0,$$
$$i = 0, 1, \cdots, N-1. \qquad (2)$$

In (2), the vector function $f(x_i, u_i, i, t_f) = f'(x_i, u_i, i\Delta t)$ or $f'((x_i + (t_f/2N)f'), u_i, (i + \frac{1}{2})\Delta t)$ depending on whether the first-order or the second-order Runge–Kutta method is used in discretization. Similarly, for the fourth-order Runge–Kutta method, the relationship between f and f' can also be easily derived [9].

Manuscript received September 18, 1989; revised December 3, 1990 and July 24, 1991. Paper recommended by Past Associate Editor, R. V. Patel. This work was supported in part by the National Science Council and the National Defense Technology Foundation under Grant CS79-0210-D009-22.

The author is with the Department of Control Engineering, National Chiao Tung University, Hsinchu, Taiwan, Republic of China.

IEEE Log Number 9107230.

The optimization problem (2) is difficult to solve due to its *nonseparability* and *large dimension*. However, if we fix the t_f, (2) becomes separable. The *separability* of a large-dimension optimization problem can be exploited to develop efficient computational procedures. Therefore, like Dantzig–Wolfe or Benders' decomposition techniques [10], the *two-level* approach naturally lends itself to solving (2).

The proposed two-level algorithm begins by decomposing (2) into a *master–slave* problem. The solution of the master problem is an estimate of the minimum final time t_f which will be passed down to the slave problem. The slave problem, which is (2) with t_f fixed in the constraints and omitted in the objective function, will determine that the given t_f is less or more than the minimum t_f depending on whether the target set $T(x(t_f), t_f) = 0$ can be reached by the available control at the given t_f or not. Taking the solution obtained from the slave problem into account, the master problem will generate a better estimate of the minimum final time t_f. Then the iterative procedures of the two-level algorithm continue until convergence occurs. The advantages of this two-level algorithm are that the master problem can be easily solved by a one-dimensional *gradient method*, and the slave problem can be solved by the developed *parallel computing method* which combines *recursive quadratic programming* with the *dual method*. Furthermore, we prove that the two-level algorithm will converge to the optimal solution of (2) under some conditions. The computational steps for the slave problem are *completely decomposed*, and the needed operations are only simple arithmetic addition, subtraction, multiplication, and division. Furthermore, the computations needed for solving the master problem also consist of only simple arithmetic operations. Therefore, with slight modifications on the step-size and the convergence checks of the iterative methods, we present a *VLSI array processor based hardware computing architecture* to realize the two-level parallel computing algorithm. The corresponding *computation time complexity* in terms of the number of algorithmic iterations, additions, and multiplications as well as the communication time is also analyzed.

Based on the analyzed time complexity and the progress of VLSI technology in fabricating multipliers, adders, and communication links [11], [12], we can estimate the computation time of the algorithm from simulation results. To demonstrate the applicability of our algorithm, we have tested several kinds of practical minimum-time control problems for which either an exact solution from *Pontryagin's maximum principle* or an approximate solution reported in the literature was available. The simulated results are very satisfactory; moreover, the estimated computation time is less than 1 ms ($= 10^{-3}$ s). This strongly suggests that our algorithm is very suitable for real-time application of minimum-time control.

II. The Two-Level Algorithm and Assumptions

A. Preliminaries

Let $\Omega(t_f)$ denote the set of points $(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{s})$ that satisfy the constraints of (2) under a given t_f, where $\boldsymbol{x} = (x_0, x_1, \cdots, x_N)$, $\boldsymbol{u} = (u_0, \cdots, u_{N-1})$, $\boldsymbol{s} = (s_0, \cdots, s_N)$. It can be easily verified that

$$\min_{t_f} \left\{ t_f + \min_{x,u,s\in\Omega(t_f)} M \sum_{i=0}^{N} s_i^T s_i \right\} \tag{3}$$

is equivalent to (2) because the optimal solution of (2) must be an optimal solution of (3), and vice versa. For a given t_f, we define the slave problem of (2) as

$$\min_{x,u,s\in\Omega(t_f)} M \sum_{i=0}^{N} s_i^T s_i \tag{4}$$

which is the minimization problem within the bracket in (3). Let $(\hat{\boldsymbol{x}}(t_f), \hat{\boldsymbol{u}}(t_f), \hat{\boldsymbol{s}}(t_f))$ denote the optimal solution of the slave problem at a given t_f, then the optimal objective value of the slave problem can be expressed as $M \sum_{i=0}^{N} \hat{s}_i^T(t_f)\hat{s}_i(t_f)$ which is a function of t_f. Based on (3), the master problem of (2) is defined as

$$\min_{t_f} \left\{ t_f + M \sum_{i=0}^{N} \hat{s}_i^T(t_f)\hat{s}_i(t_f) \right\}. \tag{5}$$

Remark 1: Let $t_f^* = \min\{t_f \mid M \sum_{i=0}^{N} \hat{s}_i^T(t_f)\hat{s}_i(t_f) = 0\}$. It then denotes the solution for the minimum time of the discretized minimum-time control problem. Furthermore, t_f^* can also be expressed as $\min\{t_f \mid \sum_{i=0}^{N} \hat{s}_i^T(t_f)\hat{s}_i(t_f) = 0\}$ because M is a constant.

B. Assumptions

1) The optimal solution of (2) is unique.
2) The functions f, T, and g are three times continuously differentiable.

C. The Two-Level Algorithm and Its Convergence

Property 1: Let $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{u}}, \hat{\boldsymbol{s}}, \hat{t}_f)$ be the optimal solution of (2). Suppose $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{u}}, \hat{\boldsymbol{s}})$ is a regular point, then $\hat{\boldsymbol{x}}(t_f)$, $\hat{\boldsymbol{u}}(t_f)$, $\hat{\boldsymbol{s}}(t_f)$, and $M \sum_{i=0}^{N} \hat{s}_i^T(t_f)\hat{s}_i(t_f)$ will be twice continuously differentiable in an open interval containing $\hat{t}_f$.

Remark 2: A *regular point* is a point at which the gradient of the active constraint functions are linearly independent [13].

Property 1 has been verified in [8, p. 39] based on previous assumptions. Because of space limitations, we will not include the complete proof here; however, a rough sketch of the proof is provided in the following. Because of Assumption 1, we have $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{u}}, \hat{\boldsymbol{s}}) = (\hat{\boldsymbol{x}}(\hat{t}_f), \hat{\boldsymbol{u}}(\hat{t}_f), \hat{\boldsymbol{s}}(\hat{t}_f))$. Then, according to (4), let $(d\hat{\boldsymbol{x}}, d\hat{\boldsymbol{u}}, d\hat{\boldsymbol{s}})$ denote the deviation from $(\hat{\boldsymbol{x}}(\hat{t}_f), \hat{\boldsymbol{u}}(\hat{t}_f), \hat{\boldsymbol{s}}(\hat{t}_f))$ induced by the deviation dt_f from $\hat{t}_f$. Thus, $(d\hat{\boldsymbol{x}}, d\hat{\boldsymbol{u}}, d\hat{\boldsymbol{s}})$ can be considered as a function of dt_f. If dt_f is sufficiently small, the values of the corresponding $(d\hat{\boldsymbol{x}}, d\hat{\boldsymbol{u}}, d\hat{\boldsymbol{s}})$ can be obtained approximately from a quadratic programming problem which approximates (2) at the point $(\hat{\boldsymbol{x}}(\hat{t}_f), \hat{\boldsymbol{u}}(\hat{t}_f), \hat{\boldsymbol{s}}(\hat{t}_f), \hat{t}_f)$. This quadratic programming problem has a positive definite Hessian matrix with dt_f as the given driving function, $(d\boldsymbol{x}, d\boldsymbol{u}, d\boldsymbol{s})$ as the minimizing variables, and $(d\hat{\boldsymbol{x}}, d\hat{\boldsymbol{u}}, d\hat{\boldsymbol{s}})$ as the optimal solution. By Assumption 1, if $dt_f = 0$, the corresponding $(d\hat{\boldsymbol{x}}, d\hat{\boldsymbol{u}}, d\hat{\boldsymbol{s}}) = 0$. Thus, we may justify Property 1 by showing the twice continuous

differentiability of $(d\hat{x}, d\hat{u}, d\hat{s})$ at $dt_f = 0$. To proceed with the proof, we first show the existence of a solution for the *Lagrange first-order necessary conditions* of the quadratic programming problem when $dt_f = 0$. This solution is $(d\hat{x}, d\hat{u}, d\hat{s}) = 0$. Secondly, we show that the Jacobian of the left-hand-side functions of the Lagrange first-order necessary conditions with respect to $(d\hat{x}, d\hat{u}, d\hat{s})$ is nonsingular at $(d\hat{x}, d\hat{u}, d\hat{s}) = 0$ based on the positive definiteness of the Hessian matrix of the quadratic programming problem and $(\hat{x}(\hat{t}_f), \hat{u}(\hat{t}_f), \hat{s}(\hat{t}_f))$ $(= (\hat{x}, \hat{u}, \hat{s}))$ being a regular point. Finally, we apply the implicit function theorem to complete the proof by showing that $(d\hat{x}, d\hat{u}, d\hat{s})$ is twice continuously differentiable at $dt_f = 0$ if Assumption 2 holds.

Since the existence of $(d/dt_f)M\sum_{i=0}^{N}\hat{s}_i^T(t_f)\hat{s}_i(t_f)$ is justified by Property 1, the master problem can be solved iteratively by the following one-dimensional gradient method:

$$t_f(j+1) = t_f(j) - \alpha\left[1 + \frac{d}{dt_f}M\sum_{i=0}^{N}\hat{s}_i^T(t_f(j))\hat{s}_i(t_f(j))\right] \tag{6}$$

where the derivative $(d/dt_f)M\sum_{i=0}^{N}\hat{s}_i^T(t_f(j))\hat{s}_i(t_f(j))$ in each iteration j can be calculated from the solution of the slave problem as described later in Section III, and the α $(0 < \alpha \leq 1)$ is a constant step-size parameter. Thus, the structure of this two-level algorithm is shown in Fig. 1, and the detailed algorithm procedures are described below.

Step 0G: Pick up $t_f(0)$, N, M and set $j = 0$.

Step 1G: Solve the slave problem, and output the value of $(d/dt_f)M\sum_{i=0}^{N}\hat{s}_i^T(t_f(j))\hat{s}_i(t_f(j))$ to the master problem.

Step 2G: If $|1 + (d/dt_f)M\sum_{i=0}^{N}\hat{s}_i^T(t_f(j))\hat{s}_i(t_f(j))| < \epsilon_2$ (a preselected accuracy), stop and output the optimal control $\hat{u}(t_f(j))$ from the solution of the slave problem; otherwise, go to Step 3G.

Step 3G: Compute $t_f(j+1) = t_f(j) - \alpha[1 + (d/dt_f)M\sum_{i=0}^{N}\hat{s}_i^T(t_f(j))\hat{s}_i(t_f(j))]$, $0 < \alpha \leq 1$.

Step 4G: Set $j = j + 1$ and return to Step 1G.

Remark 3: The notation G at the end of each step is used to indicate that the master problem is solved by the gradient method in the above two-level algorithm.

Sufficient conditions required for the convergence of the two-level algorithm are stated below.

Theorem 1: Suppose α is small enough and $t_f(0)$ is sufficiently close to $\hat{t}_f$, then the sequence $\{(\hat{x}(t_f(j)), \hat{u}(t_f(j)), \hat{s}(t_f(j)), t_f(j));\ j = 0, 1, \cdots\}$ generated from the two-level algorithm will converge to $(\hat{x}, \hat{u}, \hat{s}, \hat{t}_f)$.

The proof of Theorem 1 partially appeared in [7] and its complete details can be found in [8]. However, we will highlight the procedures of the proof in the following. Because $(\hat{x}, \hat{u}, \hat{s}) = (\hat{x}(\hat{t}_f), \hat{u}(\hat{t}_f), \hat{s}(\hat{t}_f))$, it is enough to show Theorem 1 by showing that the sequence $\{t_f(j)\}$ converges to $\hat{t}_f$. Let $c(t_f) = t_f + M\sum_{i=0}^{N}\hat{s}_i^T(t_f)\hat{s}_i(t_f)$. First, we derived in [8, Lemma 4] an important property of nonnegative second derivative which implies that there exists an $\epsilon > 0$ such that $\forall\ t_f \in (\hat{t}_f - \epsilon, \hat{t}_f + \epsilon)$, $(d/dt_f)c(t_f) < 0$, if $t_f < \hat{t}_f$, and $(d/dt_f)c(t_f) > 0$, if $t_f > \hat{t}_f$, however, $(d/dt_f)c(\hat{t}_f) = 0$. This property also indicates that the solution set of the two-level algorithm consists of a single point

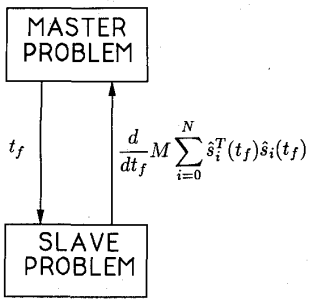

Fig. 1. The structure of the two-level algorithm.

only. Then, we prove the convergence by showing the satisfaction of three sufficient conditions of the global convergence theorem (GCT) [13, p. 187]. Based on the property of nonnegative second derivative, using Taylor's theorem, we show that the sequence $\{t_f(j)\}$ generated by our algorithm is contracting with respect to $\hat{t}_f$ provided that $t_f(0)$ is sufficiently closed to $\hat{t}_f$ and α is small enough. Therefore, *the sequence* $\{t_f(j)\}$ *lies in a compact set*, and thus the first condition of GCT is satisfied. Based on the property of nonnegative second derivative, we also show that $c(t_f(j+1)) < c(t_f(j))$ if $(d/dt_f)c(t_f) \neq 0$, and $c(t_f(j+1)) = c(t_f(j))$ if $(d/dt_f)c(t_f) = 0$. Since $c(t_f)$ is the objective function of the considered problem, this indicates that *every iteration of our algorithm reduces the objective value before the solution is obtained.* Hence, the second condition of GCT is satisfied. For the last condition of GCT, it is easy to show that our algorithm is a *closed mapping* because α is a constant.

D. Processing of Initial Guess

As indicated in Theorem 1, the initial guess $t_f(0)$ being close enough to $\hat{t}_f$ is one of the sufficient conditions for the convergence of the two-level algorithm. Basically, no prior knowledge of $\hat{t}_f$ is available; however, $\hat{t}_f$ is close to t_f^* which is defined in Remark 1. Thus, $t_f(0)$ can be chosen near t_f^* and, consequently, is close to $\hat{t}_f$. According to the definition of t_f^*, we may characterize it as follows: $\sum_{i=0}^{N}\hat{s}_i^T(t_f)\hat{s}_i(t_f) > 0$, if $t_f < t_f^*$; $\sum_{i=0}^{N}\hat{s}_i^T(t_f)\hat{s}_i(t_f) = 0$, if $t_f = t_f^*$; $\sum_{i=0}^{N}\hat{s}_i^T(t_f)\hat{s}_i(t_f) \geq 0$, if $t_f > t_f^*$. Thus, for a suitable tolerance $\epsilon_1 > 0$, if $t_f(0) < t_f^*$ and $\sum_{i=0}^{N}\hat{s}_i^T(t_f(0))\hat{s}_i(t_f(0)) < \epsilon_1$, we may consider that $t_f(0)$ is close enough to t_f^* as required. To obtain such $t_f(0)$, we may employ Newton's method to solve the single-variable nonlinear equation $M\sum_{i=0}^{N}\hat{s}_i^T(t_f)\hat{s}_i(t_f) = 0$ as the master problem in the two-level algorithm. The initial guess $\tilde{t}_f(0)$ of this two-level algorithm based on Newton's method should be less and can be much less than t_f^* as evidenced by our numerical examples. The two-level algorithm based on Newton's method will stop when $M\sum_{i=0}^{N}\hat{s}_i^T(t_f)\hat{s}_i(t_f) < M\epsilon_1$ (i.e., $\sum_{i=0}^{N}\hat{s}_i^T(t_f)\hat{s}_i(t_f) < \epsilon_1$) is satisfied, and the final value of t_f will be taken as $t_f(0)$. It is important to note that we do not intend to solve the exact t_f^* from the two-level algorithm based on Newton's method because the existence of $(d/dt_f)M\sum_{i=0}^{N}\hat{s}_i^T(t_f)\hat{s}_i(t_f)$ at $t_f = t_f^*$ is questionable under the current assumptions, and accordingly, the two-level

algorithm based on Newton's method may not converge in the large. Therefore, the tolerance ϵ_1 is an experienced value, and it should be chosen such that the two-level algorithm based on Newton's method will not diverge before $M\sum_{i=0}^{N}\hat{s}_i^T(t_f)\hat{s}_i(t_f) < M\epsilon_1$ is satisfied, and $t_f(0)$ can be close enough to t_f^*. In general, the selection of ϵ_1 is not as difficult as it seems. When the two-level algorithm based on Newton's method stops, the convergence criteria $|1 + (d/dt_f)M\sum_{i=0}^{N}\hat{s}_i^T(t_f(0))\hat{s}_i(t_f(0))| < \epsilon_2$ of the two-level algorithm based on the gradient method will be checked first. If it is satisfied, then the output $(\hat{x}(t_f(0)), \hat{u}(t_f(0)), \hat{s}(t_f(0)))$ of the slave problem is considered to be the optimal solution of (2), and this is very likely to happen for reasonably small ϵ_1. If it is not satisfied, the two-level algorithm based on the gradient method will continue the solution process from $t_f(0)$. Summarizing the above discussions, we may describe the two-level algorithm based on Newton's method for the processing of initial guess as follows.

Step 0N: Pick up $\tilde{t}_f(0)$, N, M and set $j = 0$.

Step 1N: Solve the slave problem, and output the values of $M\sum_{i=0}^{N}\hat{s}_i^T(t_f(j))\hat{s}_i(t_f(j))$ and $(d/dt_f)M\sum_{i=0}^{N}\hat{s}_i^T(t_f(j))\hat{s}_i(t_f(j))$ to the master problem.

Step 2N: If $M\sum_{i=0}^{N}\hat{s}_i^T(t_f(j))\hat{s}_i(t_f(j)) \geq M\epsilon_1$, go to Step 3N; if $M\sum_{i=0}^{N}\hat{s}_i^T(t_f(j))\hat{s}_i(t_f(j)) < M\epsilon_1$, but $|1 + (d/dt_f)M\sum_{i=0}^{N}\hat{s}_i^T(t_f(j))\hat{s}_i(t_f(j))| \geq \epsilon_2$ go to Step 1G; if $M\sum_{i=0}^{N}\hat{s}_i^T(t_f(j))\hat{s}_i(t_f(j)) < M\epsilon_1$, and $|1 + (d/dt_f)M\sum_{i=0}^{N}\hat{s}_i^T(t_f(j))\hat{s}_i(t_f(j))| < \epsilon_2$, stop and output the optimal control $\hat{u}(t_f(j))$ from the solution of the slave problem.

Step 3N: Compute $t_f(j+1) = t_f(j) - [M\sum_{i=0}^{N}\hat{s}_i^T(t_f(j))\hat{s}_i(t_f(j))/(d/dt_f)M\sum_{i=0}^{N}\hat{s}_i^T(t_f(j))\hat{s}_i(t_f(j))]$.

Step 4N: Set $j = j + 1$ and return to Step 1N.

Remark 4: The processing of initial guess accomplished by Newton's method will speed up the two-level process significantly because of the fast computational performance of Newton's method.

Remark 5: Suppose at $t_f = t_f(j)$, $(-1/(d/dt_f)\sum_{i=0}^{N}\hat{s}_i^T(t_f)\hat{s}_i(t_f)) = M' > M$, then $|1 + (d/dt_f)M'\sum_{i=0}^{N}\hat{s}_i^T(t_f)\hat{s}_i(t_f)| \leq \epsilon_2$ holds. This indicates that $(\hat{x}(t_f(j)), \hat{u}(t_f(j)), \hat{s}(t_f(j)), t_f(j))$ satisfies the convergence criteria of the two-level algorithm based on the gradient method corresponding to a larger penalty coefficient M', and hence, it is a better solution. Therefore, we may use $(-1/(d/dt_f)\sum_{i=0}^{N}\hat{s}_i^T(t_f)\hat{s}_i(t_f)) > M$ as an alternative test of convergence to replace $|1 + (d/dt_f)M\sum_{i=0}^{N}\hat{s}_i^T(t_f)\hat{s}_i(t_f)| < \epsilon_2$ in Step 2G and 2N. This alternative convergence criterion has more flexibility.

III. Parallel Computing Method for the Slave Problem

A. Preliminaries

It is well known that a large-dimension separable optimization problem can be solved efficiently by the *dual method* provided the *Hessian matrix* of the associated *Lagrangian function* is positive definite [13]. Thus, although the slave problem looks as complicated as (2), it is simpler than (2) because it is separable but (2) is not. However, for a nonlinear slave problem, the direct application of the dual method may fail due to the nonpositive definiteness of the Hessian matrix of the associated Lagrangian function. Although the *augmented Lagrangian method* [13] can guarantee the positive definiteness of the corresponding Hessian matrix, it will destroy the separability of a separable problem.

Therefore, in order to maintain separability while ensuring a positive definite Hessian matrix, we use a combination of *recursive quadratic programming* with the *dual method*.

We first convert the expression of the slave problem into a problem with equality constraints only while all the variables are within *simple bounds*. The simple bounds of a variable $(\cdot)$ are given by $(\underline{\cdot}) \leq (\cdot) \leq (\bar{\cdot})$, where $(\underline{\cdot})$ and $(\bar{\cdot})$ are the lower and upper bounds, respectively. The bound $(\underline{\cdot}) = -\infty$ if $(\cdot)$ is unbounded from below, and $(\bar{\cdot}) = +\infty$ if $(\cdot)$ is unbounded from above. Thus, any bounded or unbounded variable can be expressed within simple bounds. For the inequality constraints that cannot be expressed in the above form of variables within simple bounds, we define them as *nonsimple inequality constraints*. Let the nonsimple inequality constraints among the qN inequality constraints $g(x_i, u_i) \leq 0$, $i = 0, \cdots, N-1$ be denoted by $h'(x_i, u_i) \leq 0$, $i = 0, \cdots, N-1$, where the vector function h': $\mathbb{R}^{n+p} \to \mathbb{R}^r (r \leq q)$. These rN nonsimple inequality constraints can be converted to equality constraints by adding to them the nonnegative variables z_i, $i = 0, \cdots, N-1$, $z_i \in \mathbb{R}^r$, i.e., $h'(x_i, u_i) + z_i = 0$, $z_i \geq 0$, $i = 0, \cdots, N-1$. For notational convenience, we define the vector $y_i \equiv (x_i, u_i, z_i) \in \mathbb{R}^{n+p+r}$ and $y \equiv (y_0, y_1, \cdots, y_N)$ by taking $u_N \equiv 0$ and $z_N \equiv 0$. We also define the set $Y \equiv \{y \mid \underline{y}_i \leq y_i \leq \bar{y}_i,\ i = 0 \cdots, N\}$, where $\underline{y}_i = (\underline{x}_i, \underline{u}_i, 0)$ and $\bar{y}_i = (\bar{x}_i, \bar{u}_i, +\infty)$. Then the expression of the slave problem (4) can be rewritten as follows:

$$\min_{y \in Y, s} M\sum_{i=0}^{N} s_i^T s_i$$

subject to

$$x_{i+1} - x_i - \frac{t_f}{N} f(y_i, i, t_f) + s_i = 0, \qquad x_0 = x_o$$
$$T(x_N, t_f) + s_N = 0,\ h(y_i) = 0, \qquad i = 0, 1, \cdots, N-1 \tag{7}$$

where $h(y_i) \equiv h'(x_i, u_i) + z_i$ and $f(y_i, i, t_f) \equiv f(x_i, u_i, i, t_f)$.

B. The Recursive Quadratic Programming Method

Let $y(k) \in Y$ and let $(y(k), s(k))$ denote an estimated solution point of the slave problem in (7),where k denotes an iteration index. Moreover, we define the set $Y - y' = \{y - y' \mid y \in Y\}$. Then for any $y'' \in Y - y'$, $y'' + y' \in Y$. According to Han's work [14], (8) describes a quadratic subproblem of (7) at $(y(k), s(k))$. Let $(\boldsymbol{dy}^*(k), \boldsymbol{ds}^*(k)) \equiv (dy_0^*, \cdots, dy_N^*, ds_0^*, \cdots, ds_N^*)$ denote the solution of (8). Then $(\boldsymbol{dy}^*(k), \boldsymbol{ds}^*(k))$ is a *descent direction* of (7) at $(y(k), s(k))$ in the sense of the *absolute-value penalty function* of (7) [14], [13, p. 439].

$$\min_{dy(k) \in Y - y(k),\ ds} \sum_{i=0}^{N} \{Mds_i^T(k)ds_i(k) + 2Ms_i(k)^T ds_i(k) + \gamma dy_i^T(k)dy_i(k)\}$$

subject to

$$E_i(k) + dx_{i+1}(k) - dx_i(k) - \frac{t_f}{N} f_y(y_i(k), i, t_f) \cdot dy_i(k) + ds_i(k) = 0,$$
$$x_0(k) = x_0,\ E_N(k) + T_{x_N}(x_N(k), t_f) \cdot dx_N(k) + ds_N(k) = 0$$
$$h(y_i(k)) + h_y(y_i(k))dy_i(k) = 0, \quad i = 0, 1, \cdots, N-1 \quad (8)$$

where $E_i(k) = x_{i+1}(k) - x_i(k) - (t_f/N)f(y_i(k), i, t_f) + s_i(k)$, for $i = 0, 1, \cdots, N-1$ and $E_N(k) = T(x_N(k), t_f) + s_N(k)$; the scalar γ is a positive real number; and f_y, h_y, T_{x_N} denote the partial derivatives of f, h, and T with respect to y and x_N, respectively. Note that $f_y \equiv (f_x, f_u)$ and $h_y \equiv (h_x, h_u, h_z)$.

The recursive quadratic programming method applied to (7) is to solve (8) recursively with updating procedures

$$y(k+1) = y(k) + \rho(k)dy^*(k),$$
$$s(k+1) = s(k) + \rho(k)ds^*(k) \quad (9)$$

until convergence occurs. The step-size $\rho(k)$ in (9) is determined by the *exact line search method* to minimize the *absolute-value penalty function* of (7) while subject to $y(k+1) \in Y$ [14], [13, p. 439]. Note that once $\rho(k)$ is determined, all components of $y(k)$ and $s(k)$ can be updated in parallel.

Convergence of such a recursive quadratic programming method under some conditions has been proved by Han [14] and has also appeared in [13, p. 441]. Here, we state the relevant theorem as a lemma for our problem.

Lemma 2: [14, Theorem 3.2], [13, p. 441] Assuming that: i) there exist two positive numbers δ and ξ such that $\delta \le \min\{\gamma, M\} \le \max\{\gamma, M\} \le \xi$; and ii) there exists a unique solution $(dy^*(k), ds^*(k))$ to (8) for any $(y(k), s(k))$, and the corresponding Lagrange multiplier vector is bounded. Then, the bounded sequence $\{y(k), s(k)\}$ generated from (9) will converge to a point that satisfies the Kuhn–Tucker condition of (7).

Remark 6: The Kuhn–Tucker condition is the first-order necessary condition of an optimal solution.

Remark 7: In [14] and [13], although they only explicitly treat the problem with inequality constraints, they have indicated that the above result also applies to the problems including equality constraints.

Remark 8: It seems that there is almost no restriction on the value of γ except for positivity. However, large γ/M will cause slow convergence, while very small γ/M will induce numerical difficulties in solving (8). Thus, γ is usually chosen by experience, and a recommended value for γ is $0.02M$.

The iterative procedures of (9) are simple as long as $(dy^*(k), ds^*(k))$ is given for each k. Therefore, the difficult part of the recursive quadratic programming is to solve (8) for each iteration k. Clearly, (8) is ideally suited to the dual method because it is separable and has a positive definite Hessian matrix [13, p. 404].

C. The Dual Method

To solve (8) by the dual method, we begin by describing the dual problem of (8) below

$$\max_{\lambda} \Phi(\lambda) \quad (10)$$

where the dual function

$$\Phi(\lambda) = \min_{dy \in Y - y(k),\ ds} \sum_{i=0}^{N} \{Z_i(dy_i, ds_i) + \lambda_{f,i}^T F_i(dx_{i+1}, dy_i, ds_i) + \lambda_{h,i}^T H_i(dy_i)\} \quad (11)$$

in which

$$Z_i(dy_i, ds_i) = Mds_i^T ds_i + 2Ms_i(k)^T ds_i + \gamma dy_i^T dy_i;$$
$$H_i(dy_i) = h(y_i(k)) + h_y(y_i(k))dy_i;$$

and

$$F_i(dx_{i+1}, dy_i, ds_i) = \begin{cases} E_i(k) + dx_{i+1} - dx_i - \frac{t_f}{N} f_y(y_i(k), i, t_f)dy_i + ds_i \\ \quad \text{for } 0 \le i \le N-1; \\ E_N(k) + T_{x_N}(x_N(k), t_f)dx_N + ds_N, \\ \quad \text{for } i = N \end{cases}$$

and $\lambda \equiv (\lambda_f, \lambda_h)$ is the vector of Lagrange multiplier such that $\lambda_{f,i} \in \mathbb{R}^n$ and $\lambda_{h,i} \in \mathbb{R}^r$, $i = 0, 1, \cdots, N$, and $\lambda_{h,N} \equiv 0$ because there is no $H_N(dy_N)$. Note that for notational convenience, we omit the argument k from all variables $dy(k)$ and $ds(k)$.

The dual method we employed to solve (8) uses the gradient ascent method to solve (10). Its simple iterative procedures are

$$\lambda_{f,i}^j(l+1) = \lambda_{f,i}^j(l) + \beta(l)\nabla_{\lambda_{f,i}^j}\Phi(l), \quad i = 0, \cdots, N,\ j = 1, \cdots, n$$
$$\lambda_{h,i}^j(l+1) = \lambda_{h,i}^j(l) + \beta(l)\nabla_{\lambda_{h,i}^j}\Phi(l), \quad i = 0, 1, \cdots, N-1,\ j = 1, \cdots, r \quad (12)$$

where $\lambda_{f,i}^j$ and $\lambda_{h,i}^j$ are the jth components of $\lambda_{f,i}$ and $\lambda_{h,i}$, respectively; the step-size $\beta(l)$ is obtained from the exact line search method, i.e., $\beta(l) = \arg\{\max_{\beta \ge 0} \Phi(\lambda + \beta\nabla_\lambda \Phi(l))\}$, and all components of the gradient $\nabla_\lambda \Phi(l)$ can be computed according to the following formula:

$$\nabla_{\lambda_{f,i}^j}\Phi(l) = F_i^j(\hat{dx}_{i+1}, \hat{dy}_i, \hat{ds}_i), \quad i = 0, \cdots, N,\ j = 1, \cdots, n$$
$$\nabla_{\lambda_{h,i}^j}\Phi(l) = H_i^j(\hat{dy}_i), \quad i = 0, \cdots, N-1,\ j = 1, \cdots, r \quad (13)$$

provided the minimum solution $\hat{dy}$ and $\hat{ds}$ of (11) with $\lambda = \lambda(l)$ are obtained [13]. Note that F_i^j and H_i^j in (13) denote the jth component of F_i and H_i, respectively.

Due to the separability of (8), for any given λ, (11) can be completely decomposed into $(N+1)(2n+p+r)$ independent minimization subproblems, and each subproblem can be analytically solved. Thus, the minimum solution $\hat{dy}$ and $\hat{ds}$ of (11) with $\lambda = \lambda(l)$ can be obtained from the formula in Lemma 3 by setting $\lambda = \lambda(l)$. The derivation of Lemma 3 is described in the Appendix.

Lemma 3: For a given λ, let

$$dx_i^j = \frac{1}{2\gamma}\Big[\big(-\lambda_{f,i-1}^j + W_i^j(y_i(k), i) - \sum_{\kappa=1}^{r} h_{x_i^j}^{\kappa}(y_i(k))\lambda_{h,i}^{\kappa},$$
$$i = 1,\cdots,N,\ j = 1,\cdots,n$$
$$du_i^j = \frac{1}{2\gamma}\Big[\sum_{\kappa=1}^{n}\frac{t_f}{N} f_{u_i^j}^{\kappa}(y_i(k), i)\lambda_{f,i}^{\kappa} - \sum_{\kappa=1}^{r} h_{u_i^j}^{\kappa}(y_i(k))\lambda_{h,i}^{\kappa}\Big],$$
$$i = 0,\cdots,N-1,\ j = 1,\cdots,p$$
$$dz_i^j = \frac{1}{2\gamma}\Big[-\sum_{\kappa=1}^{r} h_{z_i^j}^{\kappa}(y_i(k))\lambda_{h,i}^{\kappa}\Big],$$
$$i = 0,\cdots,N-1,\ j = 1,\cdots,r \quad (14)$$

where

$$W_i^j(y_i(k), i) = \begin{cases} \lambda_i^j + \sum_{\kappa=1}^{n}\frac{t_f}{N} f_{x_i^j}^{\kappa}(y_i(k), i, t_f)\lambda_{f,i}^{\kappa}, \\ \quad \text{for } 0 \le i \le N-1; \\ -\sum_{\kappa=1}^{n} T_{x_N^j}^{\kappa}\lambda_{f,N}^{\kappa}, \\ \quad \text{for } i = N \end{cases}$$

and $f_{(\cdot)}^{\kappa}$ and $h_{(\cdot)}^{\kappa}$ denote the κth component functions of $f_{(\cdot)}$ and $h_{(\cdot)}$, respectively.

Then the solution $(\hat{dy}_i, \hat{ds}_i) \equiv (\hat{dx}_i, \hat{du}_i, \hat{dz}_i, \hat{ds}_i)$, $i = 0,\cdots,N$ of (11) are

$$\hat{dx}_i^j = \begin{cases} \bar{x}_i^j - x_i^j(k), & \text{if } x_i^j(k) + dx_i^j > \bar{x}_i^j; \\ dx_i^j, & \text{if } \underline{x}_i^j \le x_i^j(k) + dx_i^j \le \bar{x}_i^j; \\ \underline{x}_i^j - x_i^j(k), & \text{if } x_i^j(k) + dx_i^j < \underline{x}_i^j \end{cases} \quad (15)$$

and the $\hat{du}_i^j$ and $\hat{dz}_i^j$ are similarly obtained from the same formula in (15) by just replacing the dx by du and dz, respectively, however, $\hat{dx}_i^j \equiv 0$, $\hat{du}_N^j \equiv 0$, and $\hat{dz}_N^j \equiv 0$ for each j; Moreover,

$$\hat{ds}_i^j = -\frac{\lambda_{f,i}^j + 2M}{2M}, \qquad i = 0, 1,\cdots,N,\ j = 1,\cdots,n. \quad (16)$$

The above dual method will converge to the solution $(dy^*(k), ds^*(k))$ needed in (9). Thus, we have successfully combined the recursive quadratic programming method with the dual method to solve (7), which is the slave problem. An advantage of this combination is that the computations of (9), (12)–(16) are fully parallel except for the determination of the step-size $\rho(k)$ and $\beta(l)$.

D. The Parallel Computing Method

Following are the details of the algorithmic steps which will replace Steps 1G and 1N in the two-level algorithm.

Step 1.0: Select $(y(0), s(0))$, and set $k = 0$.

Step 1.1: Compute in parallel $E_i(k)$, $h(y_i(k))$, $(t_f/N)f_x(y_i(k), i, t_f)$, $(t_f/N)f_u(y_i(k), i, t_f)$, $h_x(y_i(k))$, $h_u(y_i(k))$, $h_z(y_i(k))$, $i = 0,\cdots,N-1$, and $E_N(k)$, $T_{x_N}(x_N(k), t_f)$.

Step 1.2: Select $\lambda(0)$, and set $l = 0$.

Step 1.3: Compute in parallel $\hat{dx}_i^j$, $\hat{du}_i^j$, $\hat{dz}_i^j$, and $\hat{ds}_i^j$, $\forall$ i, j from (14), (15), and (16) with $\lambda = \lambda(l)$.

Step 1.4: Compute in parallel $\nabla_{\lambda_{f,i}^j}\Phi(l)$ and $\nabla_{\lambda_{h,i}^j}\Phi(l)$, $\forall$ i, j from (13). Then if $|\nabla_\lambda\Phi(l)|_\infty < \eta$ (a preselected accuracy), go to Step 1.6; otherwise, go to Step 1.5.

Step 1.5: Determine the step-size $\beta(l)$ and update in parallel all components of $\lambda(l+1)$ according to (12), and set $l = l + 1$, then return to Step 1.3.

Step 1.6: Determine the step-size $\rho(k)$ and update in parallel all components of $(y(k+1), s(k+1))$ according to (9). If $\max(|\hat{dy}(k)|_\infty, |\hat{ds}(k)|_\infty) < \eta$, set $(y(t_f), s(t_f)) = (y(k), s(k))$, $(y(0), s(0)) = (y(k), s(k))$, $k = 0$, and go to Step 1.7; otherwise, set $\lambda(0) = \lambda(l)$, $l = 0$, $k = k + 1$, and return to Step 1.3.

Step 1.7: Compute $-\lambda_i^T(l)(\partial/\partial t_f)[w(y_i(k), i, t_f)]$ and $M\hat{s}_i^T(k)\hat{s}_i(k)$ and go to Step 1.8.

Step 1.8: Perform the summation $\sum_{i=0}^{N} -\lambda_i^T(t_f)(\partial/\partial t_f)w(y_i(k), i, t_f)$ and $M\sum_{i=0}^{N}\hat{s}_i^T(k)\hat{s}_i(k)$, and go to Steps 2N or 2G.

Remark 9: From the sensitivity theorem in [13, p. 313], we see that the negative value of a Lagrange multiplier associated with an equality constraint can be interpreted as the incremental rate of change in the value of the objective function per unit change in that equality constraint requirement. Furthermore, the derivative of a constraint function with respect to its parameter can also be interpreted as the incremental rate of change in the value of the constraint requirement per unit change of that parameter. Therefore, using the chain rule, $(d/dt_f)M\sum_{i=0}^{N}\hat{s}_i(t_f)^T\hat{s}_i(t_f)$ can be calculated by $\sum_{i=0}^{N} -\lambda_i^T(t_f)(\partial/\partial t_f)w(y_i(t_f), i, t_f)$, where

$$w(y_i(t_f), i, t_f) = \begin{cases} -\frac{t_f}{N} f(y_i(t_f), i, t_f), & \text{for } 0 \le i \le N-1, \\ T_{x_N}(x_N, t_f), & \text{for } i = N \end{cases}$$

and the values of $y(t_f)$, $\lambda(t_f)$ are the convergent solution and the corresponding Lagrange multiplier of the slave problem under a given t_f.

Remark 10: The notation $|(\cdot)|_\infty$ denotes the value of the largest magnitude of the components in $(\cdot)$.

Remark 11: Normalizing (8) by dividing all the terms in its objective function by M, the solution of the resultant normalized problem will be the same as (8). However, the associated Lagrange multiplier will be scaled by $1/M$.

Nonetheless, the values of the objective function of (8) and its derivative with respect to t_f can be obtained by multiplying M by the corresponding values in the normalized problem. Thus, this normalization technique will not add an extra computation load, but will speed up the convergence [8, p. 40].

Henceforth, we will call the two-level algorithm with the above parallel computing method as the two-level parallel computing algorithm.

E. Convergence Analysis

Based on the *duality theorem* [13, p. 399], if the minimum solution of (8) exists, the convergence of the dual method is well established. Indeed, the minimum solution of (8) always exists because of the slack variables, and is unique because of the positive definite Hessian matrix. Moreover, the corresponding Lagrange multiplier at the solution must be finite since the optimal objective value of (8) is always bounded. Thus, condition ii) of Lemma 2 is completely satisfied. Furthermore, condition i) in Lemma 2 is obviously satisfied because M and γ are constant positive real values. Thus, the developed parallel computing method will converge to a point that satisfies the Kuhn–Tucker condition of (7) which is the slave problem. Therefore, we have the following theorem.

Theorem 4: Any bounded sequence $\{y(k), s(k)\}$ generated from the parallel computing method will converge to a point that satisfies the Kuhn–Tucker condition of the slave problem.

It has been indicated in Section II-C that because of Assumption 1, $(\hat{x}(\hat{t}_f), \hat{u}(\hat{t}_f), \hat{s}(\hat{t}_f)) = (\hat{x}, \hat{u}, \hat{s})$ and is unique. Thus, the convergent point achieved above when $t_f = \hat{t}_f$ must be $(\hat{x}(\hat{t}_f), \hat{u}(\hat{t}_f), \hat{s}(\hat{t}_f))$ due to the uniqueness. Therefore, combining Theorem 4 with Theorem 1 shows the convergence of the two-level parallel computing algorithm.

IV. Hardware Computing Architecture and Time Complexity

A. Preliminaries

It can be observed from the two-level parallel computing algorithm that (9), (12)–(16) consist of almost all the computations needed in the algorithm. The computations in each equation are composed of independent sets of arithmetic operations. Each set corresponds to calculating the value of a single component in one time interval, for example, the calculation of dx_i^j in (14). Based on such a complete-decomposition property, it is possible to use VLSI array processors to realize the algorithm. For example, we may assign $N + 1$ processors to carry out the computations in one algorithmic step, such that the ith processor only takes care of the computations corresponding to the time interval i. However, it seems that there are difficulties in performing the summations $M \sum_{i=0}^{N} \hat{s}_i^T(t_f)\hat{s}_i(t_f)$ and $(d/dt_f)M \sum_{i=0}^{N} \hat{s}_i^T(t_f)\hat{s}_i(t_f)$ (i.e., $\sum_{i=0}^{N} - \lambda_i^T(t_f)(\partial/\partial t_f)w(y_i(t_f), i, t_f)$) in the slave problem, and passing the results to the master problem. In fact, this can be solved by using $\log_2(N + 1)$ stages of processors in between the master and the slave problems. These processors will work as two-input adders in the upward direction to perform the summations. Furthermore, they will serve as registers in the downward direction to propagate the computed value in the master problem to the slave problem. As a matter of fact, the true obstacles are the determination of the step-size and the convergence checks in Steps 1.4, 1.5, and 1.6. For example, to determine $\beta(l)$ in Step 1.5, we should input the data $\nabla_{\lambda_{f,i}}\Phi(l), \nabla_{\lambda_{h,i}}\Phi(l)$, $i = 0, \cdots, N$ computed in $N + 1$ processors to a single processor to perform the exact line search method. Such data transfer requires very complicated communication techniques under the consideration of dedicated VLSI array processors. For convergence checks, we take the determination of $|\nabla_\lambda \Phi(l)|_\infty < \eta$ in Step 1.4, for example, in which, we have to determine whether $\max\{|\nabla_{\lambda_{f,i}}\Phi(l)|_\infty, |\nabla_{\lambda_{h,i}}\Phi(l)|_\infty\} < \eta$, $0 \le i \le N$. This may require $\log_2(N + 1)$ stages of two-input logical AND gates and communication links to determine the convergence. Thus, it will consume much communication overhead and destroy the regularity of the hardware architecture.

To cope with the above two difficulties, we make two algorithmic modifications as follows.

B. Two Algorithmic Modifications

In general, a constant step-size $\hat{\beta}$ is acceptable to the gradient ascent method. Also, the employment of $\rho(k) = 1$ in the recursive quadratic programming method was justified by Powell in [15]. Moreover, the above choice of $\rho(k)$ does not violate the requirement that $y(k + 1) \in Y$ for each k. Therefore, these constant step-sizes will circumvent the exact line search.

For convergence checks, it is common that a sufficiently large number of iterations is enough to ensure the convergence of a convergent iterative method. Therefore, we may assign two arbitrary numbers of iterations, $l_{\max}$ and $k_{\max}$, to the dual method and the recursive quadratic programming method, respectively. Then the convergence of each method will be detected by the iteration counter in the individual processor.

Based on the above modifications, we may rewrite Steps 1.4, 1.5, and 1.6 as follows.

Step 1.4(m): Compute in parallel $\nabla_{\lambda_{f,i}^j}\Phi(l)$ and $\nabla_{\lambda_{h,i}^j}\Phi(l)$, $\forall\ i, j$, from (13). Then, if $l = l_{\max}$, go to Step 1.6(m); otherwise, go to Step 1.5(m).

Step 1.5(m): Update in parallel all components of $\lambda(l + 1)$ according to (12) with $\beta(l) = \hat{\beta}$, and set $l = l + 1$, then return to Step 1.3.

Step 1.6(m): Update in parallel all components of $(y(k + 1), s(k + 1))$ according to (9) with $\rho(k) = 1$. Then, if $k = k_{\max}$, set $(y(t_f), s(t_f)) = (y(k), s(k))$, $(y(0), s(0)) = (y(k), s(k))$, $k = 0$, and go to Step 1.7; otherwise, set $\lambda(0) = \lambda(l)$, $l = 0$, $k = k + 1$, and return to Step 1.3.

C. Characteristics of the Processing Elements

Before presenting the architecture of the dedicated VLSI array processors for our algorithm, we first describe the characteristics of the eight types of processing elements needed.

Table I shows the characteristics of each processing element (PE). In the first column, we indicate the type and the

corresponding time interval of a PE by its superscript and subscript, respectively, except that the $\log_2(N+1)$ stages of the PE^7 processors and the single PE^8 do not correspond to any particular time interval. Moreover, for the sake of simplicity but without losing generality, we restrict $1 \le i \le N-1$ in the table. The second column lists the corresponding algorithmic step of each PE. However, each PE except PE^7 and PE^8 will only take care of the computations of one time interval. For example, PE_2^3 will compute $\nabla_{\lambda_{f,2}^j}\Phi$, $j = 1,\cdots,n$ and $\nabla_{\lambda_{h,2}^j}\Phi$, $j = 1,\cdots,r$. The third column shows the output data of each PE. The output data of a PE are its computed data. Thus, the output data of PE_2^3 are $\nabla_{\lambda_{f,2}^j}\Phi$, $j = 1,\cdots,n$ and $\nabla_{\lambda_{h,2}^j}\Phi$, $j = 1,\cdots,r$. The fourth column indicates the destinations of the output data of each PE. As can be seen from the algorithm, the computation of each algorithmic step may require data computed from previous steps. Therefore, the description of the data flow as shown in column 4 is necessary for the construction of the architecture of VLSI array processors. Apparently from column 2, Step 0N, Step 1.0, and Step 1.2 concerning the initial guesses are not associated with any PE. However, PE_i^1 and PE_i^2 will output commands to PE_i^5 and PE_i^4 to request the data of initial guess, respectively, and PE_i^5 and PE_i^4 will respond upon request. These actions will take care of Steps 1.0 and 1.2. In Step 0N, the constants M and N should be built in the PE's which need these constants for computations. However, the value of $\tilde{t}_f(0)$ will be supplied from PE^8 once the algorithm starts execution. Furthermore, it is seen from column 4 row 2 of Table I that the output data of PE_i^2 will output to PE_i^5 if $l = l_{\max}$ is detected in PE_i^3. This implies that the PE_i^3 should send a command of convergence to PE_i^2 when $l = l_{\max}$, and PE_i^2 will respond by sending data to PE_i^5. Similar situations occur for PE_i^5 and PE^8. PE_i^5 will command PE_i^4 to send data to PE_i^6 if the convergence of the recursive quadratic programming method, i.e., $k = k_{\max}$ is detected. Furthermore, when PE^8 detects convergence of the two-level algorithm, it will command PE_i^5 array processors to output the optimal control solution through the command path $PE^8 \rightarrow PE^7$s $\rightarrow PE_i^1$, $i = 0,\cdots,N \rightarrow PE_i^5$, $i = 0,\cdots,N$. Therefore, we have the output command and the associated destinations of each PE shown in columns 5 and 6, respectively. Column 7 corresponds to the time complexity of each PE; this will be explained later.

D. VLSI Array Processors Architecture

Based on the characteristics of the PE's shown in Table I, Fig. 2 shows an overall *data-driven computing* architecture to realize the two-level parallel computing algorithm. For the sake of clarity, we let $N = 3$. Each square block in Fig. 2 denotes a PE. It should be noted that the PE's lying in the same array will perform the same algorithmic step. The structure is very regular, modular, and locally interconnected. Therefore, to implement it by VLSI array processors would be beneficial.

D.1 Communication Links: Each directed link in Fig. 2 is associated with an asynchronous handshaking communication link. The arrows indicate the direction of the data or command flow. The directed *solid links* denote the data transfer path. Therefore, the *data-driven computation* means that the computations in each PE begin after the completion of all the data transfer from solid links. The directed *dashed links* denote the command path for requesting initial guess or notifying convergence. The directed *dash-dotted links* also denote the data transfer path. They differ from the solid links because the receiving PE's will not use the transfer data for computation immediately.

D.2 Major Functions: According to row 2 of Table I, the PE^2, PE^3, and PE^4 array processors constitute the dual method. The recursive quadratic programming method is formed by the PE^1 and PE^5 array processors along with the dual method. Furthermore, the PE^6 array processors, the pyramid-like $\log_2(N+1)$ stage PE^7 array processors, and the single PE^8 together with the recursive quadratic programming method form the iterative two-level algorithm.

D.3 Synchronization: The local synchronization concerning the operation within each PE is controlled by the self-timed clock in the PE. However, the computations in the PE's lying in the same array in Fig. 2 will be carried out asynchronously simultaneously due to the asynchronous handshaking communication link and the data-driven computation. Thus, the presented overall computing architecture will achieve the parallel computation and avoid the necessity of a global reference clock which has several drawbacks [12].

E. The Realization and Time Complexity of Each PE

Basically, each PE consists of a self-timed clock, control logic unit, counter(s), and a dedicated arithmetic unit. The typical structure of a PE is shown in Fig. 3. The self-timed clock is used to control the synchronization of the operations within the PE. The dedicated arithmetic unit may consist of multipliers, adders, various types of registers, and/or some simple combinatorial logic. The registers include read only registers, read/write registers, and general-purpose registers. The read only registers are used to store some algorithm constants such as $M, \gamma, \alpha, \hat{\beta}, \epsilon_1, \epsilon_2$, etc. The read/write registers are used to store some values which will last for a while before being replaced, such as $(t_f/N)f_x(y_i(k), i, t_f)$, $(t_f/N)f_v(y_i(k), i, t_f)$ and $h_x(y_i(k))$, $h_u(y_i(k))$, $h_z(y_i(k))$ etc.; while the general-purpose registers are used to store the temporary data after each arithmetic operation in the dedicated arithmetic unit. Counter #1 in Fig. 3 is used to count the clock pulses in order to indicate the completion of the arithmetic operations. The functions of the control logic unit include the control of the sequence of arithmetic operations and the timing of activating the right communication link for sending out the data, and the reactions to the input command. However, counter #2 is available only in PE_i^3 and PE_i^5 for each i to detect whether $l = l_{\max}$ and $k = k_{\max}$ in the dual method and the recursive quadratic programming method, respectively.

According to column 2 of Table I and the details of the algorithmic steps, the structure of the dedicated arithmetic units of PE_i^3, PE_i^4, and PE_i^5 are similar but much simpler than PE_i^2. The arithmetic unit of PE^7 only consists of an adder and registers. Moreover, PE_i^6 as well as PE_i^1 are

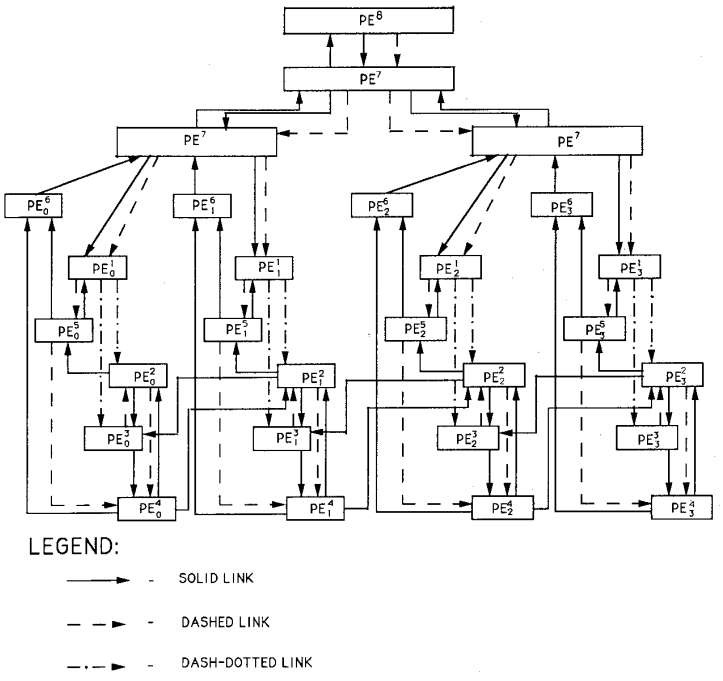

Fig. 2. The architecture of the dedicated VLSI array processors.

TABLE I
THE CHARACTERISTICS OF PE

PE_{time}^{type}	Algorithm Step	Output Data	Destination of Output Data	Output Command	Destination of Output Command	Time Complexity
PE_i^1	1.1	$(t_f/N)f_x(y_i, i, t_f)$, $(t_f/N)f_u(y_i, i, t_f)$, $h_x(y_i)$, $h_u(y_i)$, $h_z(y_i)$	PE_i^2 & PE_i^3	request initial guess	PE_i^5	unknown
PE_i^2	1.3	$\hat{dy}_i(l)$, $\hat{ds}_i(l)$	PE_i^5 if $l = l_{max}$ in PE_i^3; else, PE_i^3 & PE_{i-1}^3	request initial guess	PE_i^4	$2 \otimes$ & $[4 + \log_2(n + r + 2)] \oplus$
PE_i^3	1.4(m)	$\nabla\lambda_{f,i}\Phi(l)$, $\nabla\lambda_{h,i}\Phi(l)$	PE_i^4	convergence	PE_i^2	$1 \otimes$ & $\log_2(n + p + 3) \oplus$
PE_i^4	1.5(m)	$\lambda_{f,i}(l)$, $\lambda_{h,i}(l)$	PE_i^6 if $k = k_{max}$ in PE_i^5; else, PE_i^2 & PE_{i+1}^2	—	—	$1 \otimes$ & $1 \oplus$
PE_i^5	1.6(m)	$y_i(k)$, $s_i(k)$	PE_i^6 if $k = k_{max}$ in PE_i^5; else, PE_i^1	convergence	PE_i^4	$1 \oplus$
PE_i^6	1.7	$M\hat{s}_i^T(t_f)\hat{s}_i(t_f)$, $(d/dt_f)M\hat{s}_i^T(t_f)\hat{s}_i(t_f)$	PE^7	—	—	unknown
PE^7 s	1.8, or 4N or 4G	up: $M\sum_{i=0}^{N}\hat{s}_i^T(t_f)\hat{s}_i(t_f)$, $(d/dt_f)M\sum_{i=0}^{N}\hat{s}_i^T(t_f)\hat{s}_i(t_f)$; down: t_f	up: PE^8 down: PE_i^1	—	—	$1 \oplus$
PE^8	2N & 3N, or 2G & 3G	t_f	PE^7	convergence	PE_i^5	$4 \otimes$ & $5 \oplus$

involved with the functions of f and h which are not specified unless the system is given. Therefore, we will analyze the realization and the time complexity of PE_i^1, PE_i^2, and PE^8 explicitly, while the rest can be similarly obtained.

We start from PE_i^2 first. For the sake of simplicity, we choose $n = 3$, $p = 2$, and $r = 1$. Based on (14), (15), and (16), the details of the arithmetic unit of PE_i^2 is shown in Fig. 4(a). The square blocks denoted by AX_i^1, AX_i^2, AX_i^3, AU_i^1, AU_i^2, AZ_i^1, and AS_i correspond to the computations of dx_i^1, dx_i^2, dx_i^3, du_i^1, du_i^2, dz_i^1, and $\hat{ds}_i$, respectively, and the square blocks denoted by CC represent the constraint checkers which are used to obtain the values of $\hat{dx}_i^1$, $\hat{dx}_i^2$, $\hat{dx}_i^3$, $\hat{du}_i^1$, $\hat{du}_i^2$, and $\hat{dz}_i^1$ based on (15). From (14) and (16), it is clear that the structure of AX_i^1 is the same as AX_i^2 and AX_i^3,

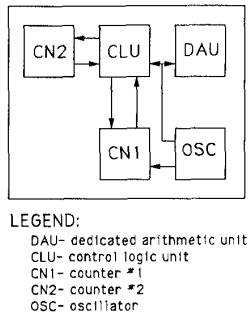

Fig. 3. A typical structure of a PE.

and is more complicated than AU_i^1, AU_i^2, AZ_i^1, and AS_i. Thus, we only show the structure of AX_i^1 in Fig. 4(b), and the rest can be similarly obtained. In Fig. 4(b), the multiplier and the adder are denoted by $\otimes$ and $\oplus$, respectively. The data $(t_f/N)f_x(y_i(k), i, t_f)$, $(t_f/N)f_u(y_i(k), i, t_f)$, $h_x(y_i(k))$, $h_u(y_i(k))$, and $h_z(y_i(k))$ input from PE_i^1 are stored in the read/write registers. The arithmetic operations have been parallelized as much as possible, and the structure is self-explanatory [according to (14)]. Concerning the structure of CC, we will only explain the one which computes $\hat{dx}_i^1$ since all the CC's in Fig. 4(a) are functionally the same. First, $x_i^1(k) + dx_i^1$ should be performed, then the sign bits of the values of $(x_i^1(k) + dx_i^1) - \bar{x}_i^1$ and $(x_i^1(k) + dx_i^1) - \underline{x}_i^1$ can be used to determine $\hat{dx}_i^1 = dx_i^1$ or $(\bar{x}_i^1 - x_i^1(k))$ or $\underline{x}_i^1 - x_i^1(k)$ by a simple 3 to 1 multiplexer. Details of the above description are shown in Fig. 4(c).

Based on Fig. 4(a) and (b), the time complexity of the longest *serial* path of computations to obtain the general dx_i^j, du_i^j, dz_i^j, and $\hat{ds}_i$ will be $2\otimes + \max[\log_2(n + r), \log_2(p + r)]\oplus$. Because $n \geq p$, in general, the above expression can be taken as $2\otimes + \log_2(n + r)\oplus$. Furthermore, the time complexity of the CC will be less than four $\oplus$ since the combinatorial logic of the 3 to 1 multiplexer is simpler than a full adder. Altogether, it takes $2\otimes$ and $[4 + \log_2(n + r)]\oplus$ to complete the arithmetic operations of PE_i^2. This time complexity is also shown in column 7 of Table I.

The time complexity for PE_i^1 is unknown (in column 7 of Table I) unless the functions f and h are specified. If f and h are polynomial functions, the way to get $(t_f/N)f_x(y_i(k), i, t_f)$, $(t_f/N)f_u(y_i(k), i, t_f)$ and $h_x(y_i(k))$, $h_u(y_i(k))$, $h_z(y_i(k))$ is similar to obtaining dx_i^j in PE_i^2. However, if they are special functions such as trigonometric functions, a look-up table built in ROM is needed to cooperate with the interpolation method to generate the trigonometric values.

In PE^8, a single bit S is used to represent the mode of computation in solving the master problem: Newton's method ($S = 0$) or the gradient method ($S = 1$). Let A and B denote the values of $M\sum_{i=0}^{N}\hat{s}_i^T(t_f)\hat{s}_i(t_f)$ and $(d/dt_f)M\sum_{i=0}^{N}\hat{s}_i^T(t_f)\hat{s}_i(t_f)$, respectively. Then the two-level algorithm converges if: i) $S = 0$, $A - M\epsilon_1 < 0$, $|1 + B| < \epsilon_2$; or ii) $S = 1$, $|1 + B| < \epsilon_2$ holds, where i) and ii) correspond to the convergence criteria of Newton's method and the gradient method, respectively. Detection of the above

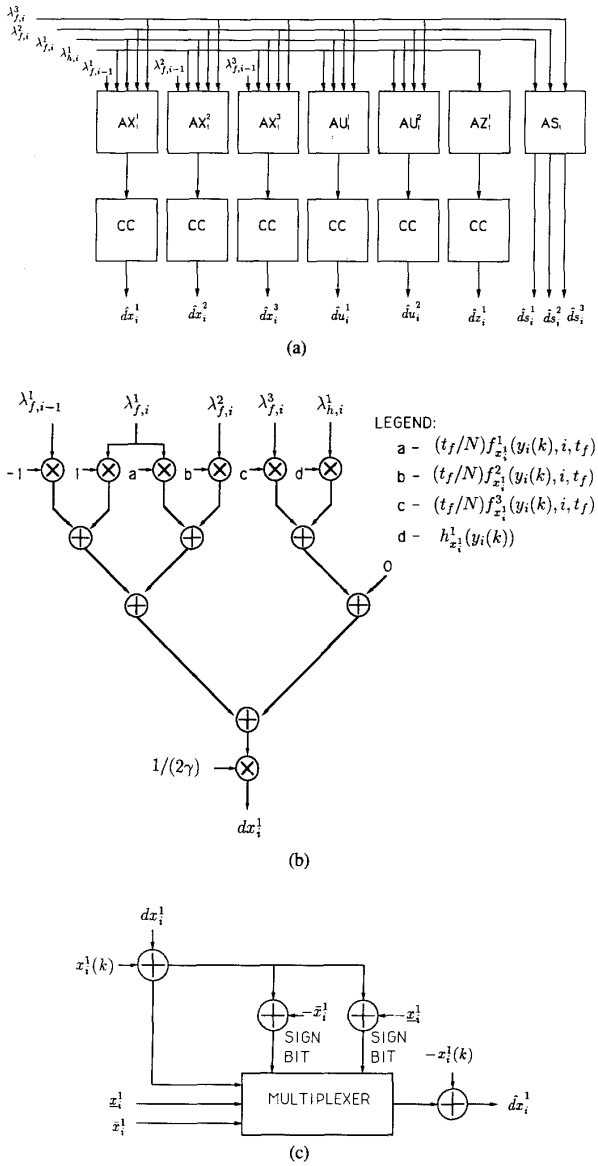

Fig. 4. (a) The structure of the dedicated arithmetic unit of PE_i^2. (b) The structure of the AX_i^1 in (a). (c) The structure of the constraint checker CC in (a).

conditions i) and ii) can be easily implemented by a simple decoder. However, if none of the above conditions holds, a calculation of the new t_f will be carried out, which is either $t_f - (A/B)$ if $S = 0$, or $t_f - \alpha(1 + B)$ if $S = 1$. Furthermore, the state of S will change from 0 to 1 if $S = 0$, $A - M\epsilon_1 < 0$, and $|1 + B| \geq \epsilon_2$ is true. Again, detection of the above condition can also be implemented by a decoder. Taking all possible parallelization into account, the time complexity of the above design for PE^8 takes around $4\otimes$ and $5\oplus$.

Finally, the time complexity corresponding to $PE_i^3 - PE^7$ can be similarly analyzed. The values are also shown in column 7 of Table I.

F. Overall Time Complexity

The two-level parallel computing algorithm consists of three nested loops. They are the iterative two-level algorithm, the recursive quadratic programming method, and the dual method from the outer to the inner loops, respectively. Let m_s denote the actual number of iterations that the iterative two-level algorithm takes to converge. Then the total number of iterations of the recursive quadratic programming method and the dual method required by the complete process are $m_s \cdot k_{\max}$ and $m_s \cdot k_{\max} \cdot l_{\max}$, respectively. The time complexity of the array PE's should only count as that of one PE since they are executed asynchronously simultaneously. Let T_{PE^j} denote the time complexity of PE^j, which is shown in column 7 and row j of Table I in terms of numbers of $\otimes$ and $\oplus$. Moreover, we let T_{CL} denote the time complexity of the asynchronous handshaking communication link which is equal to 3 clock pulses according to the design in [12, p. 347]. Similarly, the time complexity of the array communication links should just count as one T_{CL}. Then, based on the above notations and the computing architecture in Fig. 2, the overall computation time complexity is derived as shown below.

$$\begin{aligned} & m_s \cdot k_{\max} \cdot l_{\max}\left[T_{\mathrm{PE}^2} + T_{\mathrm{PE}^3} + T_{\mathrm{PE}^4} + 3T_{CL}\right] \\ & \quad + m_s \cdot k_{\max}\left[T_{\mathrm{PE}^1} + T_{\mathrm{PE}^5} + 3T_{CL}\right] \\ & \quad + m_s\left[2(\log_2 N + 1)(T_{\mathrm{PE}^7} + T_{CL}) \right. \\ & \quad \left. + T_{\mathrm{PE}^6} + T_{\mathrm{PE}^8} + 5T_{CL}\right]. \end{aligned} \tag{17}$$

Usually, the communication overhead due to the asynchronous effect of the communication links is negligible, and hence is not included in (17).

At the current stage, we have not yet developed an analysis of the convergence rate of the complete two-level algorithm. However, the linear convergence of the dual method is well known; and based on our numerical experience, the recursive quadratic programming method also converges linearly, while the mixed Newton's and gradient methods in solving the master problem converge almost quadratically. Furthermore, it is worth noting that $k_{\max}$ and $l_{\max}$ relate to the number of discretized time intervals, N, linearly.

V. Simulations

Three minimum-time control problems are described in this section. These examples are: 1) a problem with an inequality constraint on a function of the state variables; 2) a problem with simple control inequality constraints; and 3) a problem with simple inequality constraints on the control and state variables, also with an equality constraint on a function of the state variables.

Referring to the work of Sharma *et al.* [11], $T_{\otimes} = 6.75$ ns for a 16×16 bit multiplication, $T_{\oplus} \le 0.35$ ns for an addition, and the period of a clock pulse equal to 67 ps were reported. We may calculate that $T_{PE^2} = [14.9 + 0.35\log_2(n + r + 2)]$ ns, $T_{PE^3} = [6.75 + 0.35\log_2(n + p + 3)]$ ns, $T_{PE^4} = 7.1$ ns, $T_{PE^5} = 0.35$ ns, $T_{PE^7} = 0.35$ ns, $T_{PE^8} = 28.75$ ns according to column 7 in Table I, and $T_{CL} = 0.2$ ns. Then (17) becomes

$$\begin{aligned} & m_s k_{\max} l_{\max}\left[29.35 + 0.35\log_2\left((n + r + 2)(n + p + 3)\right)\right] \\ & \quad + m_s k_{\max}(T_{PE^1} + 0.95) \\ & \quad + m_s\left[1.1\log_2(N + 1) + T_{PE^6} + 29.75\right] \text{ ns}. \end{aligned} \tag{18}$$

For any given system, the functions f and h are already specified, and the dimensions n, p, and r are also given. Therefore, we may estimate the computation time of the algorithm in each example from (18) as long as the values of N, m_s, $k_{\max}$, and $l_{\max}$ are known from the simulated results.

For all examples, the second order Runge–Kutta method is used in discretization. However, the transformed parameter optimization problem is explicitly expressed only in Example 1 but neglected for the rest. Moreover, some of the algorithm constants have been set the same for all three examples; they are $\gamma = 0.02M$, $\alpha = 0.005$, $\rho = 1$, $\beta = 0.01$, $\epsilon_1 = 0.0005$, $\epsilon_2 = 0.0001$, where M is the penalty coefficient used for the problem under consideration.

Example 1. The Classical Brachistochrone Problem:

The System [1, p. 81]: Consider a bead slide on a frictionless wire in a constant gravity field as shown in Fig. 5. Its state equations are

$$\dot{x}^1 = V\cos\theta,\ \dot{x}^2 = V\sin\theta,\ \dot{V} = g\sin\theta$$

where the positions x^1, x^2 and the velocity V of the bead are the state variables, and θ, the angle of the wire with respect to the horizon, is the control variable. $g = 32$ ft/s^2 is the gravitational acceleration constant. The value of π is taken to be 3.14 rad.

The Problem: The minimum-time control considered for this system is to determine the shape of the wire which is the bead's positional trajectory so that the bead starting from $(0, 0)$ with velocity 0 will hit the line $x^1 = 1$ in minimum time while satisfying the following inequality constraint:

$$x^2 - 0.3x^1 - b \le 0$$

where b is a given real value. Three cases of b are considered, and they are: i) $b = 0.3$; ii) $b = 0.15$; and iii) $b = 0.1$, respectively.

Transformation: Using the transformation given in Section I with a second-order Runge–Kutta method for discretization, and converting the nonsimple inequality constraint to an equality constraint, the considered problem is transformed into the following:

$$\min t_f + M\sum_{i=0}^{N} s_i^T s_i$$

subject to

$$x_{i+1}^1 - x_i^1 - \frac{t_f}{N}\left[V_i + \frac{t_f}{2N}g\sin\theta_i\right]\cos\theta_i + s_i^1 = 0,\ x_0^1 = 0$$

$$x_{i+1}^2 - x_i^2 - \frac{t_f}{N}\left[V_i + \frac{t_f}{2N}g\sin\theta_i\right]\sin\theta_i + s_i^2 = 0,\ x_0^2 = 0$$

$$V_{i+1} - V_i - \frac{t_f}{N}g\sin\theta_i + s_i^3 = 0,\ V_0 = 0$$

$$x_N^1 + s_N^1 = 1,\ x_i^2 - 0.3x_i^1 - b + z_i = 0,\ z_i \ge 0,$$
$$i = 0, \cdots, N - 1.$$

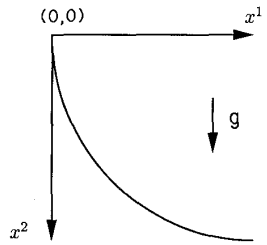

Fig. 5. The brachistochrone problem.

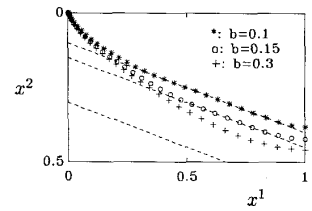

Fig. 6. Simulation result of the brachistochrone problem.

Algorithm Constants: $N = 30$, $M = 100$, $k_{\max} = 40$, $l_{\max} = 40$.

Initial Guess: We tested all three cases starting from the same initial guess which is $\tilde{t}_f(0) = 0.25$, and θ_i, $i = 0, \cdots, N-1$ and x_i^1, $i = 0, 1, \cdots, N$ are on the line segments from $\theta_0 = 90°$ to $\theta_N = 0$ and from $x_0^1 = 0$ to $x_N^1 = 1$, respectively. The rest were arbitrary.

Results and Estimated Computation Time: The test results are shown in Fig. 6. We obtained the minimum $t_f = 0.32124$, 0.32258, 0.32541 s, $m_s = 6, 7, 8$ for the three cases, respectively, and the order of the magnitudes of the final values of the slack variables was 10^{-5}. The first three of the m_s iterations were found in Newton's method, and the rest were in the gradient method. Since the state functions contain trigonometric functions, a ROM is needed to cooperate with the interpolation method to calculate the trigonometric values. The time complexity of these operations takes around $2 \otimes$ and $2 \oplus$ which makes $T_{PE^1} = 14.2$ ns. Similarly, T_{PE^6} is $4 \otimes + (2 + \log_2 n) \oplus$ which takes 28.4 ns. Moreover, for this system, $n = 3$, $p = 1$, $r = 1$. Then according to (18), the estimated computation time for the three cases are: i) 0.26 ms; ii) 0.30 ms; iii) 0.34 ms, respectively.

Discussions: It is easily seen from Fig. 6 that the inequality constraints are clearly satisfied; in fact, the result of case i) is the same as the unconstrained problem. As we expected, the minimum t_f increased as the constraints become more restrictive. However, the analytical solution for unconstrained continuous system is $t_f = 0.31325$ s which is smaller than the minimum $t_f = 0.32124$ s of case i) by 0.00799 s. This is due to the discretizing effect which is unavoidable for most of the methods implemented on a digital computer. By solving case i) with $N = 60$, $k_{\max} = 80$, $l_{\max} = 80$, the solution comes out as $t_f = 0.31708$ s which differs from the analytical solution of the continuous system by only 0.00383 s. However, the increase of N, $k_{\max}$, and $l_{\max}$ increases the hardware cost. Compared to the next two examples, the M chosen here is much smaller because this system is very sensitive to the t_f at the solution point, as can be checked from the state equations, the velocity is greatest at the final time.

Example 2: Minimum-Time Control of an Orbiting Body:

The System [3, p. 190]: Consider an orbiting body with equations of motion described in a three-axis Cartesian coordinate system

$$\dot{x}^1 = u^1 - K_x x^2 x^3, \quad \dot{x}^2 = u^2 - K_y x^1 x^3,$$
$$\dot{x}^3 = u^3 - K_z x^1 x^2,$$
$$\dot{x}^4 = (x^1 x^7 - x^2 x^6 + x^3 x^5)/2,$$
$$\dot{x}^5 = (x^1 x^6 + x^2 x^7 - x^3 x^4)/2,$$
$$\dot{x}^6 = (-x^1 x^5 + x^2 x^4 + x^3 x^7)/2,$$
$$\dot{x}^7 = -(x^1 x^4 + x^2 x^5 + x^3 x^6)/2.$$

The states x^1, x^2, and x^3 are the angular velocities with respect to the body coordinate system whose origin is the center of mass, and the states x^4, x^5, x^6, and x^7 are the variables describing position with respect to the inertially fixed coordinate system. The control variables are denoted by u^1, u^2, and u^3, and the parameters K_x, K_y, and K_z, which are functions of the moments of inertia of the body, are taken as -0.35125, 0.86058, and -0.73000, respectively.

The Problem: We consider the problem of changing the overall altitude of the above orbiting body. It is intended to find a solution for the controls u^1, u^2, and u^3 to drive the system from an initial state of

$$x^1 = x^2 = 1/57.3 \text{ rad/s}, \; x^3 = 2/57.3 \text{ rad/s},$$
$$x^4 = 0.4, \; x^5 = x^6 = 0.8, \; x^7 = 1.6$$

to the following desired final state:

$$x^1(t_f) = 1/57.3 \text{ rad/s}, \; x^j(t_f) = 0, \; j = 2, \cdots, 6,$$
$$x^7(t_f) = 2$$

so that t_f is minimum and the constraints on control

$$|u^j| \le 0.412/57.3 \text{ rad/s}^2, \qquad j = 1, 2, 3$$

are satisfied. This problem is modified from a minimum-fuel expenditure problem treated by Dyer and McReynolds [3, p. 190].

Algorithm Constants: $N = 40$, $M = 1000$, $k_{\max} = 50$, $l_{\max} = 50$.

Initial Guess: We initially set $\tilde{t}_f(0) = 16$ s and $u_i^j = 0$, $j = 1, 2, 3$, $i = 0, \cdots, N-1$. The states x_i^j, $j = 1, \cdots, 7$, $i = 0, \cdots, N$ were on the line segments from x_0^j to x_N^j, $j = 1, \cdots, 7$.

Results and Estimated Computation Time: The optimal control solution is plotted in Fig. 7; and the minimum time

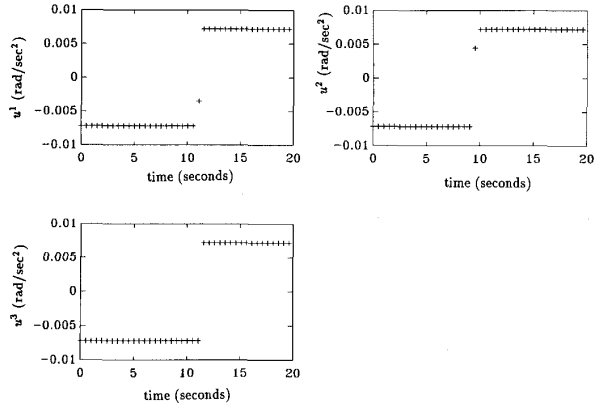

Fig. 7. Optimal control solution of the minimum-time orbit transfer problem.

obtained was $t_f = 19.5981$ s. The order of the magnitudes of the final values of the slack variables was 10^{-4}. We obtained $m_s = 2$, and they are all in Newton's method. The state function of this example is a simple function of the states and controls which takes only one multiplication to obtain in parallel the derivative $f_x(y_i(k), i, t_f)$, $f_u(y_i(k), i, t_f)$. Therefore, $T_{PE^1} = 6.75$ ns. By a similar analysis, we get T_{PE^6} as $3 \otimes + \log_2 n \oplus$ which takes 21.3 ns. Moreover, for this system, $n = 7$, $p = 3$, $r = 0$. Then by (18), the estimated computation time is 0.16 ms.

Discussions: According to Pontryagin's maximum principle, the optimal control solution of this example should be the ideal bang-bang type of control. However, the solution obtained as shown in Fig. 7 is not an ideal one because u^1 and u^2 do not abruptly change to their maximum values at their respective switching moments. The switching of u^1 and u^2 is delayed by one time interval as shown in Fig. 7. Nonetheless, our solution will get close to the ideal one provided N is large enough. This is justified by the tests of the example with $N = 50$, 60, 70, and 80 associated with appropriate values of $k_{\max}, l_{\max}$, because our test results show that the value of each control component changing to its maximum is at most delayed by one time interval which becomes smaller as N gets larger. One of the advantages of our method is that we do not assume prior knowledge of the bang-bang control solution.

Example 3: Minimum-Time Control of Robot Arm with Path Constraints:

The System: Fig. 8 shows a two-degrees-of-freedom robot arm which was presented in [16]. It consists of a revolute and a prismatic joint. Its rotating fixture is with moment of inertia J_θ through which slides a uniformly dense rod of length L_r and mass M_r. The payload has mass M_p and moment of inertia J_p, and its center of mass is at the point (x^1, x^2) which is L_p units of length from the end of the sliding rod.

Suppose that the robot operates on a horizontal plane, then there will be no gravitational loading and the following dynamic equations are easily derived based on [16] and Lagrange's equation [17, p. 158].

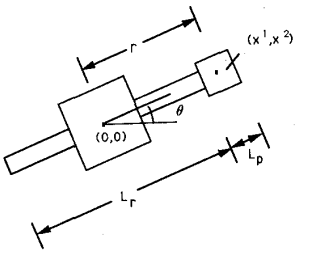

Fig. 8. The two-degree of freedom robot arm.

$$\dot{\theta} = \omega,$$
$$\dot{\omega} = \left(-k_\theta \omega + K\omega v - 2M_t \omega r v + u^\theta\right)/\left(J_t - Kr + M_t r^2\right),$$
$$\dot{r} = v,\ \dot{v} = \left(-k_r v - 0.5K\omega^2 + M_t r\omega^2 + u^r\right)/M_t$$

where r, v, θ, and ω are the state variables, and u^θ and u^r are the control variables representing the force and torque on the r and θ joints, respectively; k_θ and k_r are the friction coefficients of θ and r joints, respectively, and the constants $M_t = M_r + M_p$, $K = M_r(L_r + 2L_p)$, $J_t = J_\theta + J_p + M_r(L_p^2 + L_r L_p + L_r^2/3)$.

The Problem: It is intended to find the optimal control u^θ and u^r so that the robot will move from $(1, 1)$ to $(1, -1)$ along the straight line connecting these two points in minimum time while satisfying the control constraints that $|u^j| \le 1$, $j = \theta, r$. Note that the problem assigns a path constraint which is $r = \sec\theta$, $-(\pi/4) \le \theta \le \pi/4$ in terms of polar coordinates.

Algorithm Constants: $N = 30$, $M = 1000$, $k_{\max} = 40$, $l_{\max} = 40$.

Initial Guess: Initially, we set the following values: $\tilde{t}_f(0) = 5.5$ s, θ_i, $i = 0, \cdots, N$ and r_i, $i = 0, \cdots, N$ were on the line segments from $\theta_0 = \pi/4$ to $\theta_N = -\pi/4$ and from $r_0 = 1.414$ to $r_N = 1.414$, respectively; $\omega_i = 0$, $i = 0, \cdots, N$, $u_i^r = u_i^\theta = 0$, $i = 0, \cdots, N-1$.

Results and Estimated Computation Time: The solution of minimum time we obtained was $t_f = 6.2014$ s; the order of the magnitudes of the final values of the slack variables was 10^{-4}; the optimal trajectories of θ and ω are shown in Fig. 9(a) and (b), respectively. We obtained $m_s = 2$, and they are all in Newton's method. Since the state functions are trigonometric; the time complexity $T_{PE^1} = 14.23$ ns and $T_{PE^6} = 28.4$ ns are obtained. Furthermore, for this system, $n = 4$, $p = 2$, $r = 1$. Then according to (18), the estimated computation time is 0.091 ms.

Discussion: Fig. 10 is an optimal θ versus optimal ω plot obtained from Fig. 9(a) and (b). Although the minimum t_f was not indicated explicitly in [16], we found that the results shown in Fig. 10 agree with Fig. 8 in [16] if the θ and ω are converted to the parameters used in [16].

VI. Conclusion and Further Research

We have developed a theoretically sound, hardware implementable two-level parallel computing algorithm for *general* minimum-time control problems. This algorithm has good numerical properties as demonstrated by the simulations of a

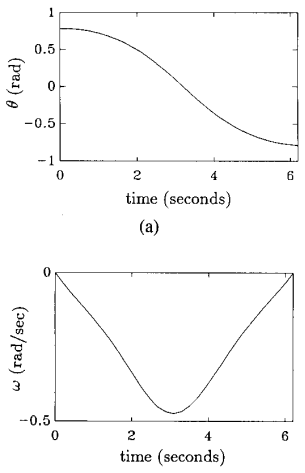

Fig. 9. (a) The optimal solution of the rotating angle $\theta(t)$. (b) The optimal solution of the rotating angular velocity $\omega(t)$.

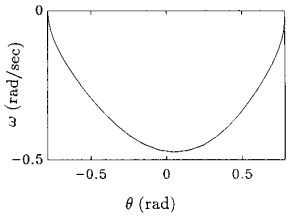

Fig. 10. The plot of optimal $\theta(t)$ versus optimal $\omega(t)$.

number of practical problems. Its realizability via VLSI array processors points to a direction of implementing the control algorithms. Its computational efficiency as evidenced by the simulated results recommends itself to real-time applications. Moreover, the parallel computing method for the slave problem can be extended to solve general optimal control problems.

However, from our numerical experience, it seems that the assumptions on the uniqueness of the optimal solution can be relaxed but a more elaborate proof is needed. Furthermore, it is worth investigating the related issue of developing a feedback scheme based upon the proposed algorithm for minimum-time control [7]. Moreover, a separate effort is needed to explore the chip size of the presented VLSI array processors for implementation.

Appendix

Proof of Lemma 3

The expression of (11) can be rearranged as

$$\min_{dy\in Y-y(k),\, ds} \sum_{i=0}^{N} \{ Z_i(dy_i, ds_i) + \mathcal{F}_i(\lambda_{i-1}, \lambda_i, dy_i, ds_i) + \lambda_{h,i}^T H_i(dy_i) \} \quad \text{(A1)}$$

where

$$\mathcal{F}_i(\lambda_{i-1}, \lambda_i, dy_i, ds_i) = \begin{cases} \lambda_i^T E_i(k) + (\lambda_{i-1} - \lambda_i)^T dx_i \\ \quad -\lambda_i^T \left[\frac{t_f}{N} f_y(y_i(k), i, t_f)\, dy_i + ds_i \right], \\ \quad \text{for } 0 \le i \le N-1, \\ \lambda_N^T E_N(k) + \lambda_{N-1}^T dx_N \\ \quad +\lambda_N^T \left[T_{x_N}(x_N(k), t_f)\, dx_N + ds_N \right], \\ \quad \text{for } i = N. \end{cases}$$

Clearly, (A1) can be decomposed into $N+1$ subproblems with each subproblem i shown below

$$\min_{dy_i \in [\underline{y}_i - y(k),\, \bar{y}_i - y(k)],\, ds_i} Z_i(dy_i, ds_i) + \mathcal{F}_i(\lambda_{i-1}, \lambda_i, dy_i, ds_i) + \lambda_{h,i}^T H_i(dy_i). \quad \text{(A2)}$$

Each term in (A2) is linear in (dy_i, ds_i) except $Z_i(dy_i, ds_i)$. Moreover, all the quadratic terms in $Z(dy_i, ds_i)$ are decoupled. Therefore, (A2) can be further decomposed into $(2n + p + r)$ minimization subproblems, and the solution of each component of $(\hat{dy}, \hat{ds})$ shown in Lemma 3 is the solution of the corresponding decomposed subproblem. This completes the proof of Lemma 3.

Acknowledgment

The reviewers' constructive comments, Prof. R. V. Patel's effort in improving the readability, and the helpful suggestions from M. H. Cheng on the presentation of this paper are deeply appreciated by the author.

References

[1] A. E. Bryson and Y. C. Ho, *Applied Optimal Control.* Waltham, MA: Blaisdell, 1969.
[2] A. P. Sage and C. C. White, *Optimum Systems Control*, 2nd ed. Englewood Cliffs, NJ: Prentice Hall, 1977.
[3] P. Dyer and S. R. McReynolds, *The Computation and Theory of Optimal Control.* New York: Academic, 1970.
[4] D. H. Jacobson and D. Q. Mayne, *Differential Dynamic Programming.* New York: American Elsevier, 1970.
[5] J. Vlassenbroeck and R. V. Dooren, "A Chebyshev technique for solving nonlinear optimal control problems," *IEEE Trans. Automat. Contr.*, vol. 33, pp. 333-340, Apr. 1988.
[6] J. E. Cuthrell and L. T. Biegler, "On the optimization of differential-algebraic process systems," *AIChE J.*, vol. 33, Aug. 1987.
[7] S. Y. Lin, "A two-level computational algorithm with parallel computing capability for general minimum-time control problems," in *Proc. Amer. Contr. Conf.*, San Diego, CA, May 23-25, 1990, pp. 2583-2588.
[8] S. Y. Lin, "A new algorithm for minimum-time control," *Dept. Contr. Eng.*, National Chiao Tung Univ., Hsinchu, Taiwan, ROC, Tech. Rep. TR-MIST-F79006, July 1990.
[9] K. E. Atkinson, *An Introduction to Numerical Analysis*, 2nd ed. Singapore: Wiley, 1989.
[10] L. S. Lasdon, *Optimization Theory for Large Systems.* New York: Macmillan, 1970.
[11] R. Sharma, A. Lopaz, J. Michejda, S. Hillenius, J. Andrews, and A. Studwell, "A 6.75-ns 16 × 16-bit multiplier in single-level-

metal CMOS technology," *IEEE J. Solid State Circuits*, vol. 24, pp. 922–927, Aug. 1989.
[12] S. Y. Kung, *VLSI Array Processors*. Englewood Cliffs, NJ: Prentice Hall International, 1988.
[13] D. G. Luenberger, *Linear and Nonlinear Programming*, 2nd ed. Reading, MA: Addison-Wesley, 1984.
[14] S. P. Han, "A globally convergent method for nonlinear programming," *J. Optimiz. Theory Appl.*, vol. 22, no. 3, pp. 297–309, July 1977.
[15] M. J. D. Powell, "Algorithms for nonlinear constraints that use Lagrangian functions," *Math. Programming*, vol. 14, pp. 224–248, 1978.
[16] K. G. Shin and N. D. McKay, "Minimum time control of robotic manipulators with geometric path constraints," *IEEE Trans. Automat. Contr.*, vol. AC-30, June 1985.
[17] R. P. Paul, *Robot Manipulators: Mathematics, Programming and Control*. Cambridge, MA: M.I.T. Press, 1981.

Shin-Yeu Lin was born in Taiwan, ROC. He received the B.S. degree in electronics engineering from National Chiao Tung University, the M.S. degree in electrical engineering from the University of Texas at El Paso, and the D.Sc. degree in systems science and mathematics from Washington University, St. Louis, MO, in 1975, 1979, and 1983, respectively.

From 1984 to 1985, he was with Washington University working as a Research Associate then a Visiting Assistant Professor. From 1985 to 1986, he joined the GTE Laboratories working at the Switching Department as a Senior MTS. He is currently an Associate Professor at the National Chiao Tung University, Taiwan, ROC. His major research interests include optimal control, optimization theory and applications, parallel and distributed computations, and large-scale power systems.