Abstract—We present a VLSI-array-processor architecture for the implementation of a nonlinear programming algorithm that solves discrete-time optimal control problems for nonlinear systems with control constraints. We incorporate this hardware module with a two-phase parallel computing method and develop a VLSI-array-processor architecture to implement a receding horizon controller for constrained nonlinear systems. On the basis of current VLSI technologies, the estimated computing time to obtain the receding-horizon feedback-control solution meets the real-time processing-system needs. © 1997 Elsevier Science Ltd.

1. Introduction

Discrete-time optimal control problems for nonlinear systems with control constraints are popular control problems. Numerous numerical techniques (Dyer and McReynolds, 1970) have been developed for solving this type of problem; however, computational efficiency is still a major issue that is frequently encountered in the solution methods. In this paper, we shall propose an algorithm that combines a projected Jacobi method and a Lagrange-dual Jacobi method to solve discrete-time optimization problems for nonlinear systems, which include discrete-time optimal control problems with control constraints. Our algorithm will achieve computational efficiency by using a VLSI array processors to meet real-time processing-system needs. Thus the first contribution of this paper is the implementation of our nonlinear programming algorithm.

Implementing a computational algorithm by VLSI array processors to improve the computational efficiency has been a trend, especially in the area of signal processing; for example, Frantzeskakis and Liu (1994) deal with a least-squares problem with linear equality constraints. However, the nonlinear programming problem considered in this paper is more complicated because of the presence of nonlinearity and inequality constraints. As well as applications to solving discrete-time optimal control problems for nonlinear systems, our hardware-implementable algorithm has important applications to receding-horizon control. In recent years, there has been a growing interest for the design of receding-horizon feedback controllers (Mayne and Michalska, 1990a; Clarke and Scattolini, 1991; Mayne and Polak, 1993; Richalet, 1993; DeNicolao et al., 1996). For such controllers, stability is guaranteed for the zero-terminal-state strategy (Mosca and Zhang 1992; DeNicolao and Scattolini, 1994). The most distinguished characteristic of this controller compared with other control methodologies is its global stability for general nonlinear systems, as shown by Mayne and Michalska (1990b, 1991); however, this is at the expense of high computational complexity to obtain a control solution. Although model reduction is an attractive approach to reduce the computational complexity (Richalet, 1993), this approach may not apply to a general, especially a highly nonlinear, system. Thus, to cope with this computational difficulty, we have proposed a two-phase parallel computing method (Lin, 1993, 1994) to obtain the solution for receding-horizon feedback control. The phase 1 method uses a two-level (master- and slave-level) approach to solve a feasibility problem to obtain an admissible control and horizon pair. The control solution obtained in phase 1 is improved by phase 2, and the final solution is taken as the receding-horizon feedback control solution for the current sampling interval. The problems formulated in this two-phase method, except for the master-level problem in phase 1, are discrete-time optimization problems for nonlinear systems. Thus we can use our hardware-implementation algorithm as a basic algorithm module in the two-phase method, and this results in a simpler algorithm than that in Lin (1994) for solving a receding-horizon feedback control solution. Since the master problem in phase 1 can be solved by simple calculations, this suggests that the two-phase method can be implemented by VLSI array processors. Thus presenting a VLSI-array-processor architecture for a receding-horizon feedback controller is the second contribution of this paper.

Since receding-horizon control is one of the most promising globally stabilizing control methodologies for highly nonlinear systems, the work described in this paper also represents an effort to realize a real-time controller for nonlinear systems.

2. Basic hardware module for a nonlinear programming algorithm

2.1. Statement of the the discrete-time optimization problem of a nonlinear system. We consider discrete-time optimization problems for nonlinear systems of the form

\[
\min_{y_t} \sum_{i=0}^{N-1} y_t M_{yi},
\]

where \( y_{t+1} = K y_t + p(y_t), \) \( y_t \in \mathbb{R}^n, \) \( i = 0, \ldots, N - 1, \)

\[
\psi(y_t) = 0, \quad y_t \in [\bar{y}_t, \bar{y}_t], \quad i = 0, \ldots, N,
\]

where the \( n \times n \) matrix \( M_t \) is positive-semidefinite, \( y = (y_0, \ldots, y_N), \) \( y_t \in \mathbb{R}^n, \) \( i = 0, \ldots, N, \) are variables to be solved. \( K \) is an \( n \times n \) constant matrix, \( p(y_t) \) is a \( n \)-dimensional vector function of \( y_t, \) \( \psi(y_t) = 0 \) represents the terminal constraints and \( \psi \) is a \( q \)-dimensional vector function.

For \( i = 0, \ldots, N, \) \( \bar{y}_t \) and \( \bar{y}_t \) represent the upper and lower bound respectively of the variables \( y_t. \) The discrete-time optimal control problem...
for a nonlinear system with quadratic objective function and simple control constraints is a case of (1) under the conditions that

\[ y_i = \langle x_i, y_i \rangle, \quad K = \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix}, \quad p(y) = \begin{bmatrix} f(x_i, x_i) \\ 0 \end{bmatrix}, \]

\[ \bar{y}_i = \frac{x_i}{x_i}, \quad y_i = \begin{bmatrix} x_i \\ 0 \end{bmatrix}, \]

where \( I \) denotes the identity matrix and 0 denote the zero submatrix or zero vector function with appropriate dimensions; \( x_i = x_i \cdot f(x_i, u_i) = 0 \) is the system dynamic equation. Note that the formulation in (1) include the case of general control constraints; for the sake of simplicity, it is not discussed here, but the explanation can be found in Section 3.

2.2. The algorithm. To solve (1), we propose an algorithm that combines the projected Jacobi method with the Lagrange-dual Jacobi method. The projected Jacobi method uses the following iterations:

\[ y_i(t + 1) = y_i(t) + \gamma_i \delta y_i(t), \quad i = 0, \ldots, N, \]  

(2)

where \( \gamma_i \) is a step size and \( l \) is the iteration index. Here \( \delta y_i(t) \) is the solution of the following quadratic subproblem:

\[ \min_{y} \sum_{i=0}^{N} \left[ M_i y_i + [M_i y_i]^T d y_i \right] 
+ \lambda_i \left[ E_i(t) + d y_{i+1} - K y_{i+1} + p_i(y_i) \right], \]

\[ \bar{y}_i = \frac{y_i}{y_i}, \quad y_i = \begin{bmatrix} x_i \\ 0 \end{bmatrix}, \]

where the dual function \( \phi(y) = \min_{y \in R^n} \sum_{i=0}^{N} \left[ M_i y_i + [M_i y_i]^T d y_i \right] 
+ \lambda_i \left[ E_i(t) + d y_{i+1} - K y_{i+1} + p_i(y_i) \right], \]

\[ \lambda_i \left[ E_i(t) + d y_{i+1} - K y_{i+1} + p_i(y_i) \right] \]

is a function of the Lagrange multiplier \( \lambda_i \in R^n, i = 0, \ldots, N \). We can use the following Lagrange-dual Jacobi method to solve (4):

\[ \lambda(t + 1) = \lambda(t) + \gamma \Delta \lambda(t), \quad i = 0, \ldots, N, \]  

(0)

where \( \gamma_i \) is a step size, and \( \Delta \lambda(t) = \left( \Delta \lambda_0(t), \ldots, \Delta \lambda_N(t) \right) \) is obtained by solving the linear equations

\[ \text{diag} \left[ \overline{V}_{\lambda_i} \phi''(A(t)) \right] \Delta \lambda(t) + \overline{V}_i \phi(A(t)) = 0. \]  

(7)

Here \( \phi''(A) \), the unconstrained dual function of \( \phi \), is defined by relaxing the inequality constraints on primal variables of \( \phi \) as follows:

\[ \phi''(A) = \min_{x} \sum_{i=0}^{N} \left[ d y_i^T \bar{M}_i d y_i + [M_i y_i]^T d y_i \right] 
+ \lambda_i \left[ E_i(t) + d y_{i+1} - K y_{i+1} + p_i(y_i) \right], \]

\[ d y_i^T \bar{M}_i d y_i + [M_i y_i]^T d y_i \]  

where \( \overline{V}_{\lambda_i} \phi''(A) \) is the Hessian matrix of \( \phi''(A) \), diag \[ \overline{V}_{\lambda_i} \phi''(A) \] is a diagonal matrix whose diagonal elements are taken from the diagonal elements of \( \overline{V}_i \phi(A) \), and \( \overline{V}_i \phi(A) \) is the gradient of \( \phi \) with respect to \( A \). Based on Luenberger (1984), \( \overline{V}_i \phi(A) \) and \( \overline{V}_{\lambda_i} \phi''(A) \) can be computed by

\[ \overline{V}_i \phi(A(t)) = \begin{cases} E_i(t) + \bar{y}_{i+1}(A(t)) - K \bar{y}_i + p_i(y_i) \bar{y}_i(t), & i = 0, \ldots, N - 1, \vspace{12pt} \\
\phi(y_i(t)) + p_i(y_i(t)) \bar{y}_i(t), & i = N. \end{cases} \]  

(9)

\[ \text{diag} \left[ \overline{V}_{\lambda_i} \phi''(A(t)) \right] = \begin{cases} -\text{diag} \left[ \left( -K + p_i(y_i(t)) \right) \bar{M}_i \right] & -K + p_i(y_i(t)) \bar{M}_i, \quad i = 0, \ldots, N - 1, \\
-\text{diag} \left[ p_i(y_i(t)) \bar{M}_i \right], & i = N. \end{cases} \]  

(10)

where \( \phi(y_i(t)) \in R^n, i = 0, \ldots, N, \) in (9) is the solution of the constrained minimization problem on the right-hand side of (5) with \( A = \lambda(t) \). Since \( \overline{V}_i \phi''(A(t)) \) is a diagonal matrix, we can compute \( \Delta \lambda(t) \) from (7) analytically by

\[ \Delta \lambda(t) = -\text{diag} \left[ \overline{V}_{\lambda_i} \phi''(A(t)) \right] \overline{V}_i \phi(A(t)), \quad i = 0, \ldots, N. \]  

(11)

It can easily be verified from (10) that \( \text{diag} \left[ \overline{V}_{\lambda_i} \phi''(A(t)) \right] \) is negative-definite. Thus \( \Delta \lambda(t) \), \( i = 0, \ldots, N, \) obtained from (11) is an ascent direction of (4). Then the Lagrange-dual Jacobi method will converge and solve (4) provided that \( \gamma_i \) is small.

However, to calculate \( \overline{V}_i \phi(A(t)) \), we need the value of \( \delta y_i(t) \), \( i = 0, \ldots, N \), which can be found by the following two steps. First, we solve for the solution \( \delta y_i, i = 0, \ldots, N \), of the unconstrained minimization problem on the right-hand side of (8), which can be obtained analytically by

\[ \delta y_i(t) = \begin{cases} \bar{y}_i - \gamma \bar{y}_i \left( \frac{y_i(t) + \delta y_i(t) - \gamma y_i(t)}{\gamma y_i(t)} \right), & \gamma y_i(t) \
\bar{y}_i - \gamma \bar{y}_i \left( \frac{y_i(t) + \gamma y_i(t) - \gamma y_i(t)}{\gamma y_i(t)} \right), & \gamma y_i(t) \end{cases} \]  

(12)

Then, we project \( \delta y_i, i = 0, \ldots, N \), to the constraint set \( y_i(t) + \gamma y_i(t) = \gamma y_i(t) \), and the resulting projection \( \delta y_i, i = 0, \ldots, N \), can be obtained analytically by

\[ \delta y_i(t) = \begin{cases} \bar{y}_i - \gamma \bar{y}_i \left( \frac{y_i(t) + \delta y_i(t)}{\gamma y_i(t)} \right), & \gamma y_i(t) \
\bar{y}_i - \gamma \bar{y}_i \left( \frac{y_i(t) + \gamma y_i(t)}{\gamma y_i(t)} \right), & \gamma y_i(t) \end{cases} \]  

(13)

\[ i = 0, \ldots, N, \]  

It can easily be verified that \( \delta y_i, i = 0, \ldots, N, \) obtained from (12) and (13) are indeed the solutions of the constrained minimization problem on the right-hand side of (5).
2.2.2. The algorithm steps.

Step 1. Initially guess the value of \( y(0) \) and set \( I = 0 \).

Step 2. Compute \( F_i(t), \) \( p_i(t), i = 0, \ldots, N - 1, \) \( \phi_{yi}(t), \) and \( \psi_{yi}(t). \)

Step 3. Initially guess the value of \( \lambda(0), \) or set it as the previous value, and set \( I = 0. \)

Step 4. Solve \( \lambda_i(t), i = 0, \ldots, N, \) by (12) and (13).

Step 5. Compute \( V_i, \phi(A(t))(t) \) (9), and \( \text{diag} \{ V_i, \phi(A(t)) \} \) by (10), \( i = 0, \ldots, N. \)

Step 6. Compute \( \lambda_i(t + 1) = \lambda_i(t) - \gamma_i \{ \text{diag} \{ V_i, \phi(A(t)) \} \}^{-1} V_i, \phi(A(t))(t), i = 0, \ldots, N. \) Check whether \( ||\Delta \lambda_i|| < \varepsilon; \) if yes, go to Step 7; otherwise, set \( I = I + 1 \) and return to Step 4.

Step 7. Compute \( \lambda_0(t + 1) = \lambda_0(t) + \gamma_0 \Delta y_{i}(t), i = 0, \ldots, N. \) Check whether \( ||\Delta y_i|| < \varepsilon; \) if yes, stop; otherwise, set \( I = I + 1 \) and return to Step 6.

2.3. The VLSI-array-processor architecture for implementing the algorithm. Since both \( M_t \) and \( \text{diag} \{ V_i, \phi(A(t)) \} \) are diagonal matrices, \( M_t^{-1} \) and \( \text{diag} \{ V_i, \phi(A(t)) \}^{-1} \) can be computed analytically. Thus all the computations required in our algorithm steps are simple arithmetic operations, and are independent with each other on different time intervals owing to the complete decomposition property. This motivates us to implement the proposed algorithm using VLSI array processors by assigning a processing element (PE) to the computation required in a time interval of an algorithm step.

2.3.1. Modification of the convergence criteria. Since our algorithm converges, the \( \Delta \lambda(t) \) in Step 6 and the \( \Delta y_i(t) \) in Step 7 will approach zero as the number of iterations \( I \) and \( t \) increase. Thus, instead of using a tolerance of accuracy, \( \varepsilon, \) for convergence criteria in Steps 6 and 7, we may assign an arbitrary number of iterations \( I_{\text{max}} \) for the Lagrange-dual Jacobi method and \( t_{\text{max}} \) for the projected Jacobi method, and modify the convergence criteria in Steps 6 and 7 as follows.

Step 6(m). \( \ldots \) If \( I \geq I_{\text{max}} \) go to Step 7; \( \ldots \)

Step 7(m). \( \ldots \) If \( I \geq I_{\text{max}} \) stop; \( \ldots \)

2.3.2. The mapping of the algorithm steps to the VLSI-array-processor architecture. Suppose we assign one PE for performing the computation of an algorithm step in a time interval; all the PEs should be linked so that the data and command flows in between PEs can make the PE arrays perform the algorithm just as in a sequential computer. Thus the construction of the array-processor architecture should be based on the data and command flow in the algorithm steps. Examples of data flows are as follows: the data \( \Delta y_{i}(t) \) and \( \Delta y_{i+1}(t) \) computed in Step 4 are needed in the computation of \( V_i, \phi(A(t)) \) in Step 5; the data \( V_i, \phi(A(t)) \) and \( V_i, \phi(A(t)) \) computed in Step 5 are needed in the computation of \( \lambda_i(t + 1) \) in Step 6(m). The command flow is more complicated; for example, in Step 6(m), if the Lagrange-dual Jacobi method converges, the data \( \Delta y_{i}(t) \) computed in Step 2 should be transferred to Step 7(m). This is a procedure of data flow followed by a command flow. In our algorithm, there are two types of commands: one is the initial-value request in Steps 1 and 3; another is the notification of convergence in Steps 6(m) and 7(m).

Figure 1 shows the VLSI-array-processor architecture for implementing our algorithm; for the sake of simplicity, but without loss of generality, we let \( N = 3 \) in Fig. 1. Each square block in Fig. 1 denotes a PE. The PEs lying in the same array require any computation, they will be taken care of by PE\( _1 \) and PE\( _2 \) respectively. We shall explain how the initial values are provided when we introduce columns 5 and 6 of Table 1. The arrows of these three types of links describe the data and command flows in the architecture.

In the following, we shall explain the mapping of the algorithm steps to the architecture with the aid of Table 1. In the first column of the table, we indicate the type and the corresponding time interval of a PE by superscript and subscript respectively. The second column lists the corresponding algorithm step of each PE, which means that the computations in a time interval or a logical decision for convergence check carried out in an algorithm step will be performed in the corresponding PE. For example, PE\( _1 \) will compute \( V_0, \phi(A(t)) \) and \( \text{diag} \{ V_0, \phi(A(t)) \} \). Although Steps 1 and 3 concerning the initial-value guesses do not require any computation, they will be taken care of by PE\( _1 \) and PE\( _2 \) respectively. We shall explain how the initial values are provided when we introduce columns 5 and 6 of Table 1. Thus each algorithm step has a corresponding PE. The third and fourth columns show the output data and the corresponding destination of each PE, where the output data of a PE are its computed data. These two columns explain
the mapping of the data flow in the algorithm to the architecture, as described by the following examples. The data $E_i(l)$ (or $\phi(x_i,l)$) if $i = N$ computed in PE$^1$ is sent to PE$^2$. Since these data will not be used for computation immediately, the data flows are indicated by dashed-dotted links directed from PE$^1$ to PE$^2$ and PE$^2$ to PET in Fig. 1. This corresponds to the data flow from Step 2 to Step 5. The output data $d_y(l)$ computed in PE$^2$ is sent to PET, which is indicated by solid links directed from PE$^2$ to PET in Fig. 1. This corresponds to the data flow from Step 4 to Step 5. However, when PET detects $t = t_{\text{max}}$, it will halt the execution and output the solution, as described in columns 5 and 6 of the third row in Table 1 and the dotted links directed from PET to PE in Fig. 1. Columns 5 and 6 show the output commands and corresponding destinations of each PE. There are two types of commands: one is the request for initial guesses and another is the notification of convergence. As we have described earlier, Steps 1 and 3 of each PE consists of a self-timed clock, a control logic unit, and a dedicated arithmetic unit. The typical structure of a PE is shown in Fig. 2. 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 and various types of registers. Counter #1 in Fig. 2 is used to count clock pulses in order to indicate the completion of the arithmetic operations. Counter #2 is available only in PE$^2$ and PE$^3$ for each $i$, and detects whether $t = t_{\text{max}}$ or $t = t_{\text{max}}$ in the Lagrange-dual Jacobi method and the projected Jacobi method respectively. Note that once the projected Jacobi method respectively. Note that once Counter #2 has reached the value of $t_{\text{max}}$, it will be reset for next count of clock pulse. The functions of the control logic unit include the control of the sequence of arithmetic operations and the timing of the right communication link for sending out the data and the data-driven-computation PE (Kung, 1988) associated with an asynchronous handshaking communication link (Kung, 1988) for data and command flows can be the solution. Therefore, the computation in each PE will be activated after the completion of all the data transfers from the solid links, this will ensure that the computations in PEs lying in the same array are carried out asynchronously and simultaneously. Nevertheless, a self-timed clock is needed in each PE to control the synchronization of the operations in each individual PE.

2.5. Realization of PEs and time complexity. Basically, each PE consists of a self-timed clock, a control logic unit, two counters and a dedicated arithmetic unit. The typical structure of a PE is shown in Fig. 2. 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 and various types of registers. Counter #1 in Fig. 2 is used to count clock pulses in order to indicate the completion of the arithmetic operations. Counter #2 is available only in PE$^2$ and PE$^3$ for each $i$, and detects whether $t = t_{\text{max}}$ or $t = t_{\text{max}}$ in the Lagrange-dual Jacobi method and the projected Jacobi method respectively. Note that once Counter #2 has reached the value of $t_{\text{max}}$, it will be reset for next count of clock pulse. The functions of the control logic unit include the control of the sequence of arithmetic operations and the timing of the right communication link for sending out the data and the data-driven-computation PE (Kung, 1988) associated with an asynchronous handshaking communication link (Kung, 1988) for data and command flows can be the solution. Therefore, the computation in each PE will be activated after the completion of all the data transfers from the solid links, this will ensure that the computations in PEs lying in the same array are carried out asynchronously and simultaneously. Nevertheless, a self-timed clock is needed in each PE to control the synchronization of the operations in each individual PE.

The mapping of the data flow in the algorithm to the architecture, as described by the following examples. The data $E_i(l)$ (or $\phi(x_i,l)$) if $i = N$ computed in PE$^1$ is sent to PE$^2$. Since these data will not be used for computation immediately, the data flows are indicated by dashed-dotted links directed from PE$^1$ to PE$^2$ and PE$^2$ to PET in Fig. 1. This corresponds to the data flow from Step 2 to Step 5. The output data $d_y(l)$ computed in PE$^2$ is sent to PET, which is indicated by solid links directed from PE$^2$ to PET in Fig. 1. This corresponds to the data flow from Step 4 to Step 5. However, when PET detects $t = t_{\text{max}}$, it will halt the execution and output the solution, as described in columns 5 and 6 of the third row in Table 1 and the dotted links directed from PET to PE in Fig. 1. Columns 5 and 6 show the output commands and corresponding destinations of each PE. There are two types of commands: one is the request for initial guesses and another is the notification of convergence. As we have described earlier, Steps 1 and 3 of each PE consists of a self-timed clock, a control logic unit, and a dedicated arithmetic unit. The typical structure of a PE is shown in Fig. 2. 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 and various types of registers. Counter #1 in Fig. 2 is used to count clock pulses in order to indicate the completion of the arithmetic operations. Counter #2 is available only in PE$^2$ and PE$^3$ for each $i$, and detects whether $t = t_{\text{max}}$ or $t = t_{\text{max}}$ in the Lagrange-dual Jacobi method and the projected Jacobi method respectively. Note that once Counter #2 has reached the value of $t_{\text{max}}$, it will be reset for next count of clock pulse. The functions of the control logic unit include the control of the sequence of arithmetic operations and the timing of the right communication link for sending out the data and the data-driven-computation PE (Kung, 1988) associated with an asynchronous handshaking communication link (Kung, 1988) for data and command flows can be the solution. Therefore, the computation in each PE will be activated after the completion of all the data transfers from the solid links, this will ensure that the computations in PEs lying in the same array are carried out asynchronously and simultaneously. Nevertheless, a self-timed clock is needed in each PE to control the synchronization of the operations in each individual PE.

2.5. Realization of PEs and time complexity. Basically, each PE consists of a self-timed clock, a control logic unit, two counters and a dedicated arithmetic unit. The typical structure of a PE is shown in Fig. 2. 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 and various types of registers. Counter #1 in Fig. 2 is used to count clock pulses in order to indicate the completion of the arithmetic operations. Counter #2 is available only in PE$^2$ and PE$^3$ for each $i$, and detects whether $t = t_{\text{max}}$ or $t = t_{\text{max}}$ in the Lagrange-dual Jacobi method and the projected Jacobi method respectively. Note that once Counter #2 has reached the value of $t_{\text{max}}$, it will be reset for next count of clock pulse. The functions of the control logic unit include the control of the sequence of arithmetic operations and the timing of the right communication link for sending out the data and the data-driven-computation PE (Kung, 1988) associated with an asynchronous handshaking communication link (Kung, 1988) for data and command flows can be the solution. Therefore, the computation in each PE will be activated after the completion of all the data transfers from the solid links, this will ensure that the computations in PEs lying in the same array are carried out asynchronously and simultaneously. Nevertheless, a self-timed clock is needed in each PE to control the synchronization of the operations in each individual PE.

Table I. The characteristics of PE

<table>
<thead>
<tr>
<th>PE Step</th>
<th>Algorithm output of output</th>
<th>Destination of output data</th>
<th>Output command</th>
<th>Destination of output command</th>
<th>Time complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE$^1$</td>
<td>1 2</td>
<td>$E_i(l), p_y(i)$ if $i = N$, $\phi(x_i,l), \psi_y(l)$ if $i = N$</td>
<td>PE$^2$ for $p_y(l)$, PE$^2$ for $E_i(l)$</td>
<td>Request of initial guess if $l = 0$</td>
<td>PE$^2$ unknown</td>
</tr>
<tr>
<td>PE$^2$</td>
<td>3 4</td>
<td>$d_y(A(l))$</td>
<td>PE$^2$ if $t = t_{\text{max}}$ in PE$^2$ else, PE$^2$ and PE$^3$, PE$^4$</td>
<td>Request of initial guess if $l = 0, t = 0$</td>
<td>PE$^4$ $2 \sigma$ and $\log_2 (2n + 2) \sigma$</td>
</tr>
<tr>
<td>PE$^3$</td>
<td>5</td>
<td>$\lambda_i(t)$</td>
<td>PE$^2$ and PE$^2_{t+1}$</td>
<td>Convergence if $t = t_{\text{max}}$</td>
<td>PE$^2$ $2 \sigma$ and $\log_2 (n + 1) \sigma$</td>
</tr>
<tr>
<td>PE$^4$</td>
<td>6(m)</td>
<td>$\lambda_i(t+1)$</td>
<td>PE$^2$ and PE$^3_{t+1}$</td>
<td>Halt if $l = t_{\text{max}}$ in PE$^2$, else, PE$^2$</td>
<td>PE$^2$ $1 \delta$</td>
</tr>
<tr>
<td>PE$^5$</td>
<td>7(m)</td>
<td>$y_i(l+1)$</td>
<td>PE$^2$ and PE$^3_{t+1}$</td>
<td>Halt if $l = t_{\text{max}}$ in PE$^2$, else, PE$^2$</td>
<td>PE$^2$ $1 \delta$</td>
</tr>
</tbody>
</table>

**Legend:**
- DAU - dedicated arithmetic unit
- CLU - control logic unit
- CN1 - counter #1
- CN2 - counter #2
- OSC - oscillator

Fig. 2. Typical structure of the processing elements.
reactions to the input command. For example, as shown in columns 3 and 4 of the third row of Table 1, and control logic circuitry in PEf should determine which of the following solid links should be activated based on the value of the circuitry in PE: should. determine which of the following reactions to the input command. For example, as shown in iteration index appearing in Counter #2: the solid link f(x(t), u(t)), derive the time complexity of the computations of PE by algorithm steps, the structure of the dedicated arithmetic and arithmetic units. For example, the formulas (12) and (13) for the computation of one component dyj, say dyj, in PEf can be realized as in Fig. 3, in which part of the multiplexer is used to perform the projection (13). From Fig. 3, we can derive the correct time complexity of the computations of PE by taking the greatest possible advantage of parallelism shown in columns 7 of the third row of Table 1, where Ω and Φ denote the times required for performing a multiplication and an addition respectively. The time complexities for the computations required for PEf, PE: and PEY can be obtained similarly to that of PEf. However, the time complexity of PEY cannot be analyzed unless the function p(yi) is given.

2.6. Summary of the operations of VLSI array processors. We can summarize the operations of the VLSI array processors shown in Fig. 1 as follows. The computations start from PEf, which commands PE: to send the initial value of yj, and then compute Ei(t) and pi(t), Ev(t) is sent to PEf, while pi(t) is sent to PE: After receiving pi(t), PE: commands PE: to send the initial value of λj, and then calculates dyj, which is sent to PEf and PEY, respectively. After receiving dyj and dyj+, PEf will compute vj, φ(λj(t)) and Vj, φ'(λj(t)), which are sent to PE: PE: will then compute λj(t+1) and send to PEf and PEY, respectively. The PE arrays formed by the PEf, PE: and PEY array will perform the Lagrange-dual Jacobi method until t = tmax is detected in PEf. When t = tmax, the PE: array will command the PEY array to send the data dyj(t) to the PEf array. Then the PEf array will update yi(t+1) and continue the above process until t = tmax is detected in PEf and halt the execution.

3. Application to receding-horizon controller

3.1. The implementable receding-horizon controller. For a nonlinear system with control constraints described by \( \dot{x} = f(x(t), u(t)) \), with \( x(t) \in \mathbb{R}^n \) and \( u(t) \in \mathbb{Q} \), where \( f: \mathbb{R}^n \times \mathbb{Q} \rightarrow \mathbb{R}^n \) is twice continuously differentiable and satisfies \( f(0,0) = 0 \) and \( \mathbb{Q} \) is the set of admissible controls containing a non-empty convex polytope. Mayne and Michalska (1990a, b, 1991) proposed a globally stable implementable receding-horizon controller. Their control strategy employed a hybrid system \( \dot{x} = f(x(t), u(t)) \), with \( x(t) \in \mathbb{R}^n \) and \( u(t) \in \mathbb{Q} \), where \( f = f_0(0,0) + f_1(0,0) \) is formed by applying a linear feedback control \( u = Cx \) to the linearized system in a neighborhood \( W \) with small enough radius and centered at origin, where \( C \) is the feedback gain matrix. Their algorithm first calculates an admissible control and horizon pair \( [u_{x_0}, t_{x_0}] \in \mathcal{Z}W(x_0), \)

where the initial state \( x_0 \in \mathbb{W} \) is assumed, the set \( \mathcal{Z}W(x) = \{ u \in \mathbb{S}, t \in (0, \infty) \} \times \mathbb{W} \}, \) with \( \mathbb{S} \) denotes the set of all piecewise-continuous functions, \( x(t) \) \( denotes the resulting state after applying the control \( u \) and \( \mathbb{W} \) denotes the boundary of \( W \). The algorithm then sets \( h = 0, s_x = 0, u_x = u_0, t_{x_0} = t_{x_0} \) and performs the following process repeatedly to yield the receding-horizon feedback control.

It applies the obtained control \( u_h \) for \( x \in \mathbb{W} \) and/or the linear feedback control \( Cx \) for \( x \in \mathbb{W} \) to the real system over \( [t_h, t_h + \Delta t] \), where \( \Delta t \in (0, \infty) \). Let \( x_{s+1} \) be the resulting state at \( t_{s+1} = t_h + \Delta t \); then, if \( x_{s+1} \notin \mathbb{W} \), the algorithm switches the control to \( u = Cx \) over \( [t_h, \infty) \); otherwise, it calculates an improved control and horizon \( [u_{s+1}, t_{s+1}] \) in the sense that \( [u_{s+1}, t_{s+1}] \in \mathcal{Z}W(x_{s+1}), \)

\[ V(x_{s+1}, u_{s+1}, t_{s+1}) \leq V(x_h, u_h, t_h, u_{s+1}, t_{s+1} - \Delta t) \]

where

\[ V(x, t, u, t) = \int_{t}^{t_{s+1}} \left( \frac{1}{2} \| \dot{x}(t) \|^2 + \| u(t) \|^2 \right) dt + \int_{t_{s+1}}^{t_{s+1}+\Delta t} \left( \frac{1}{2} \| x(t) - x(t_{s+1}) \|^2 + \frac{1}{2} \| u(t) \|^2 \right) dt, \]

in which \( R \) and \( Q \) are positive-definite matrices, \( \| y \|^2 \)

\( y \alpha \) denotes \( y^T A y \), \( x \) denotes the state trajectory in region \( W \) with feedback control \( u = Cx \).

The two-phase parallel computing method (Lin, 1994) aims to obtain a receding-horizon feedback control solution for every \( \Delta t \) time interval \( \Delta t \), \( t \leq t + \Delta t \), based on Mayne and Michalska's algorithm. In the first phase, we discretize the system into \( N \) time intervals and use slack variables to formulate the following feasibility problem, which can also be called the phase 1 problem, to obtain an admissible control and horizon pair as required in (14):

\[ \min \sum_{i=0}^{N} \| s_i \|, \]

\( x_{i+1} - x_i = f(x_i, u_i) + s_i, x_0 = x(0), x_N = x_N + s_N = 0, \)

\( uu + b = 0, i = 0, \ldots, N - 1, \)

where \( s \) denotes the vector of slack variables, and we explicitly express the non-empty convex polytope in \( \Omega \) by the set of \( q \)-dimensional linear inequality constraints on \( a_u + b = 0 \), where the matrix \( a \in \mathbb{R}^{\Omega \times \Omega} \) and the vector \( b \in \mathbb{R}^q \).

To apply the algorithm we proposed in Section 2, we need to

![Fig. 3. The arithmetic unit for computing a component dyj of the vector dyj in PEf.](image-url)
reformulate the inequality constraints (16d'). First, we separate the simple inequality constraints \(u_i \leq u_i \leq \bar{u}_i\), from \(au_i + b + z = 0\) and then convert the rest of the inequality constraints to equality constraints \(a' u_i + b + z_i = 0\) by adding positive variables \(z_i\), where \(a' \in \mathbb{R}^{m \times m}, r \leq q\). We can then rewrite (16d') as

\[
a'u_i + b + z_i = 0, \quad u_i \leq u_i \leq \bar{u}_i, \quad z_i \geq 0, \quad i = 0, \ldots, N - 1.
\]

(16d)

If there is no simple inequality constraint for \(u_i\), we may set \(\bar{u}_i = \infty\) and \(u_i = -\infty\).

Suppose that the optimal objective value of the phase 1 problem (16a-d) under a proper horizon \(t_f\) is zero; this \(t_f\) and control solution is then the admissible horizon and control pair required in (14). Because \(t_f\) is unspecified in (16a, d), we use a two-level (master- and slave-level) approach to solve the phase 1 problem (16a-d). The program in the master level of the two-level method is to determine a \(t_f\), which is passed to the slave level, and the slave problem is (16a-d) with a fixed \(t_f\) given by a master program. The master program is simple; it increases \(t_f\) by \(\delta t_f\) each iteration, where \(\delta t_f\) is a small positive real number. However, to increase the computational speed, we apply a gradient method for the first few iterations. Thus the master program (Lin, 1994) is as follows:

\[
\begin{align*}
\left\{ (t_f(l+1) = t_f(l) + \frac{\Delta}{\partial t_f}\sum_{i=0}^{N-1} \ell(t_f(t_i), t_i(t_f)) & > \varepsilon, \\
(t_f(l+1) = t_f(l) + \delta t_f & \quad \text{if} \quad \sum_{i=0}^{N-1} \ell(t_f(t_i), t_i(t_f)) < \varepsilon \quad \text{and} \quad \neq 0, \\
\text{stop otherwise},
\end{align*}
\]

(17)

where \(\ell(t_f)\) denotes the solution of the slack variables in the slave problem under a given \(t_f\). The slave problem is a case of the nonlinear programming problem considered in Section 2, which can be solved by the algorithm presented in Section 2.

When the master program stops, the zero objective value of the phase 1 problem is achieved. This means that the admissible control and horizon pair is obtained. Let \([a_i, t_f(t_i)]\) denote the admissible control and horizon pair obtained from phase 1 of the two-phase method; the phase 2 method will then improve \([a_i, t_f(t_i)]\) in the sense of reducing the performance index \(V(x_i, u_i, t_f)\), as required in (15). Thus, in phase 2, we shall solve the following phase 2 problem:

\[
\begin{align*}
\min_{i=0}^{N-1} \sum_{i=0}^{N-1} V(x_i, u_i + a_i, t_f(t_i) + \delta t_f) & = 0, \\
x_i + 1 - t_{f} = x(t_i), & \quad x_i = 0, \\
a'u_i + b + z_i = 0, & \quad u_i \leq u_i \leq \bar{u}_i, \\
\end{align*}
\]

(18)

where the \(0s\) in \(M_i\) and \(K\) denote respectively the zero submatrix and the zero vector with appropriate dimensions. The phase 2 problem (18) is also a special case of (1) with

\[
y_i = (x_i, u_i, z_i), \quad M_i = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \\
K = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & a & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \\
p(y_i) = \begin{bmatrix} f(x_i, u_i) \\ b \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -\psi(y_i) - x_i \\ x_i \\ x_i \\ x_i \end{bmatrix}
\]

Thus the VLSI-array-processor architecture presented in Fig. 1 can readily be used to implement the solution methods for the solution methods for the slave problem and the phase 2 problem. Although the computing formulas for these two problems are the same, the data in these formulas are different. Therefore we need to use one bit to represent the mode for these two problems. We let 0 represent solving the slave problem and 1 represent solving the phase 2 problem. Thus this one-bit mode can be used to control a multiplexer to select the corresponding data, as shown in Fig. 4 for the calculation in PEs.

What remains for the architecture to implement in the two-phase parallel computing method is the master program in phase 1. As we have shown in (17), the master program is very simple. Thus we may use a processing element PE\(_i\) to perform the formula given in (17). However, the data required in PE\(_i\) should be provided from the solution of the slave problem from all time intervals. Let \((\ell(t_f(t_i)), t_f(t_i), \bar{a}(t_f(t_i), \bar{a}(t_f(t_i)) = \text{the solution of the slave problem under a given } t_f; \text{ then we need PE}^a_i \text{ array processors to calculate the values of } \ell(t_f(t_i)) \text{ and } -\ell(t_f(t_i)) \text{ for } \ell(t_f(t_i)), \bar{a}(t_f(t_i)), \text{ and pyramid-like } \log_2(N + 1) \text{ stage PE}^a_i \text{ array processors. The PE}^a_i \text{ array processors are two-input adders in the upward direction used to form the sums } \sum_{i=0}^{N-1} \ell(t_f(t_i)) \text{ and } \sum_{i=0}^{N-1} \bar{a}(t_f(t_i)) \text{ which equals } (d/dt_f) \sum_{i=0}^{N-1} \ell(t_f(t_i)) \text{ (Lin, 1994). These are the data needed in PE}^2_i \text{ to perform the gradient method when } \sum_{i=0}^{N-1} \ell(t_f(t_i)) < \varepsilon. \text{ The PE}^2_i \text{ array processors are registers in the downward direction used to propagate the value of } t_f \text{ computed in the master program to the slave problem. Thus the overall VLSI array processors to implement the two-phase method are as shown in Fig. 5, in which the architectures of PE}^1_i, \text{ PE}^2_i, \text{ PE}^3_i, \text{ PE}^4_i, \text{ and PE}^5_i \text{ arrays are almost the same as in Fig. 1, except for the addition of dotted links directed from PE}^2_i \text{ to PE}^4_i, \text{ as explained below. Because, when the slave problem is solved for a given } t_f, \text{ the data } \ell(t_f) \text{ stored in PE}^2_i \text{ should be sent to PE}^4_i; \text{ these dotted links represent the fact that when PE}^2_i \text{ detects convergence of the slave problem, it will command PE}^4_i \text{ to send data to PE}^5_i. \text{ Therefore there also exist solid links from PE}^4_i \text{ to PE}^5_i \text{ in Fig. 5. Note that in phase 1, PE}^2_i \text{ will not halt the execution when detecting } l = l_{\text{max}}. \text{ This is different from Fig. 1.}

3.3. The operations of the VLSI array processors for the two-phase method. Initially, PE\(_i\) will provide a value of \(t_f(0)\) and set the mode to be 0, and will pass down the value of

Fig. 4. The multiplexer controlled by the problem mode for selecting the corresponding data.
Fig. 5. VLSI-array-processor architecture for the two-phase parallel computing method with $N = 3$.

**LEGEND:**
- SOLID LINK
- DOTTED LINK
- DASH-DOTTED LINK

$t_i(0)$ and mode 0 to the $PE_1$ array processors through the $PE^2$ pyramid-like array processors, as shown by the solid links directed from $PE^2$ through the $PE^3$ arrays to the $PE_1$ array in Fig. 5. Then the $PE_1$, $PE_2$, $PE^3$, $PE_4$, and $PE_5$ arrays will perform the algorithm proposed in Section 2 to solve the slave problem under the value of $t_f$ given by $PE_1$ until convergence in $PE_1$ is detected, that is, when $l = l_{\text{max}}$. The $PE_2$ array processors will then output the values of $S_i(t_f)$ and $(\partial/\partial t_f)(x_i, u_i)$ to the $PE_3$ array processors, and will command the $PE_4$ array to send the data of $\hat{x}_i(t_f)$ to $PE_5$. The $PE_4$ will compute $S_i(t_f)\hat{x}_i(t_f)$ and $\delta^2 \bar{S}_i(t_f)(\partial/\partial t_f) f(\bar{x}_i(t_f), \bar{u}_i(t_f))$, and the $PE_3$ arrays of processors will perform the sums $\sum_{i=0}^N S_i(t_f)\bar{S}_i(t_f)$ and $\sum_{i=0}^N \delta^2 \bar{S}_i(t_f)(\partial/\partial t_f)f(\bar{x}_i(t_f), \bar{u}_i(t_f))$ and input them to the $PE_5$ processor to perform (17). This process will continue until $PE_5$ detects the convergence of the phase 1 problem, that is, $\sum_{i=0}^N S_i(t_f)\bar{S}_i(t_f) = 0$; then the value of the admissible horizon $t_f$ and the command of mode changing to 1 will be passed to the $PE_1$ array processors through the $PE^3$ arrays. The mode-change command is indicated by dotted links directed from $PE_5$ through the $PE^3$ arrays to the $PE_1$ array as shown in Fig. 5. The $PE_1$ array will then command the $PE_2$, $PE^3$, $PE_4$, and $PE_5$ arrays to change the mode to 1. For clarity, we do not show in Fig. 5 the dotted links for the rest of the mode-change command that occurs among the $PE_1$, $PE_2$, $PE^3$, $PE_4$, and $PE_5$ arrays. At this point, the solution of the phase 1 method, $\hat{x}_i$, $\bar{x}_i$, $\bar{u}_i$, $i = 0, \ldots, N$, is stored in the $PE_1$ array. The $PE_1$, $PE_2$, $PE^3$, $PE_4$, and $PE_5$ arrays will then proceed to solve the phase 2 problem until $l = l_{\text{max}}$ is detected in $PE_1$, which will halt the execution and output the solution.

3.4. Overall-time complexity. From Section 3, we see that all the computations of the two-phase method lie in the Lagrange-dual Jacobi method; thus the total time complexity spent in the Lagrange-dual Jacobi method is the dominant term of the overall-time complexity. Let $m_1$ denote the actual numbers of iterations that the iterative two-level phase 1 problem takes to converge. Then the total number of iterations of the Lagrange-dual Jacobi method performed in phase 1 is $m_1 l_{\text{max}} l_{\text{max}}$. Furthermore, the total number of iterations of the Lagrange-dual Jacobi method performed in phase 2 is $l_{\text{max}} l_{\text{max}}$. The time complexity of the array PEs should count as only that of one PE, since they are executed asynchronously and simultaneously. Let $T_{\text{PE}}$ denote the time complexity of $PE_1$, which is shown in column 7 of Table 1 in terms of numbers of $\oplus$ and $\otimes$. Also, let $T_{\text{CI}}$ denote the time complexity of the asynchronous handshake communication link, which is equal to 3 clock pulses according to the design in Kung (1988). Similarly, the time complexity of the array communication links should count as just one $T_{\text{CI}}$. Thus the total time complexity spent in the Lagrange-dual Jacobi method based on the above notation and the computing architecture shown in Fig. 5 is

$$T_{\text{PE}} = \left(m_1 l_{\text{max}}^2 + l_{\text{max}}^2 m_1 + 3 T_{\text{CI}} \right). \quad (19)$$

3.5. Simulations. According to the work of Yano et al. (1990), $T_{\text{C}} = 3.8 \text{ ns}$ for a $16 \times 16$-bit multiplication, $T_{\text{C}} \leq 0.2 \text{ ns}$ for an addition, and the period of a clock pulse is approximately $40 \text{ ps}$. We may calculate that $T_{\text{PE}} = \left(7.6 + 0.2 \log_2 (2n + 2) \right) \text{ ns}$, and $T_{\text{CI}} = \left(7.6 + 0.2 \log_2 (n + 1) \right) \text{ ns}$, according to column 7 of Table 1, and $T_{\text{CI}} = 0.1 \text{ ns}$. Then (19) becomes

$$T_{\text{PE}} = \left(m_1 l_{\text{max}}^2 + l_{\text{max}}^2 m_1 \right) \times \left[23.1 + 0.2 \log_2 \left(2n + 2 \right) \right] \text{ ns}. \quad (20)$$
\[ x''' = x^a, \quad x'^a = -1, \]
\[ x'' = -x^a + [1.4 - 0.14(x^a)^2] + 4u, \quad x^a = -1, \]
(21)
where \( x^a \) and \( x'' \) are state variables and \( u \) is the scalar control. We intend to find a control solution that satisfies the instantaneous control constraints \(|u| \leq 0.7\) and that drives the system from the initial state \((-1, -1)\) at time \( t = 0 \) to \((0,0)\) asymptotically. The following initial values are assumed in the phase 1 method: \( \bar{t}_i = 5s, \quad u_i = 0, \quad 0 < i < N - 1, \) and \( x^a_i = x^a_{i-1} - (i/N)x^a, \quad x^a_i = x^a_{i-1} - (i/N)x^a, \quad i = 0, 1, \ldots, N \). The matrix
\[ Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \]
and \( R = 1 \) are used in phase 2. The linear feedback control \( u = -x^a - 2x^a \) is employed in the region \( W = \{|x^a| \leq 0.5\} \), to result in negative eigenvalues for the linearized closed-loop system at \((0,0)\). The algorithmic parameters are arbitrarily assigned to be \( N = 30, \quad e = 0.001, \quad \gamma_1 = \gamma_2 = \gamma_3 = 0.1, \quad 8t_i = 0.2s, \quad l_{max} = 40, \quad f_{max} = 40 \). Solving the example by our two-phase method-based implementable receding-horizon feedback control algorithm, we obtain \( N = 30 \) before reaching the region \( W \), and the complete state trajectory is shown in Fig. 6.

**Estimated computation time for the two-phase method.** In this example, \( n = 3 \) which is composed of two states and one control. For all \( h \), including \( h = 0 \), we have \( m_h = 1 \). The estimated computation time of the two-phase algorithm calculated from (19) is 0.08 ms. This shows that the receding-horizon controller hardware meets the real-time processing system needs.

**4. Conclusions**
We have presented the architecture of a basic hardware module to implement a nonlinear programming algorithm that solves discrete-time optimal control problems for nonlinear systems with quadratic objective function and control constraints. We have applied this basic hardware module in the two-phase method, and it results in a simpler algorithm than that in Lin (1994) for solving a receding-horizon feedback control solution. We have also presented the VLSI-array-processor architecture for this receding-horizon controller. The estimated computation time to obtain a receding-horizon feedback control solution is of the order of 0.1 ms, which meets the real-time processing requirement.

**Acknowledgement**—This research is supported by National Science Council, Taiwan under Grant NSC84-2213-E-009-132.

**References**