SIAM J. OPTIMIZATION
Vol. 1, No. 4, pp. 620–642, November 1991

012

PARTITIONED DYNAMIC PROGRAMMING FOR OPTIMAL CONTROL*

STEPHEN J. WRIGHT†

Abstract. Parallel algorithms for the solution of linear-quadratic optimal control problems are described. The algorithms are based on a straightforward decomposition of the domain of the problem, and are related to multiple shooting methods for two-point boundary value problems. Their arithmetic cost is approximately twice that of the serial dynamic programming approach; however, they have the advantage that they can be efficiently implemented on a wide variety of parallel architectures. Extension to the case in which there are box constraints on the controls is simple. The algorithms can be used to solve linear-quadratic subproblems arising from the application of Newton's method or two-metric gradient projection methods to nonlinear problems.

Key words. discrete-time optimal control, parallel algorithms, dynamic programming, multiple shooting

AMS(MOS) subject classifications. 49M40, 65Y05, 90C39

1. Introduction.

The unconstrained N-stage discrete-time optimal control problem with Bolza objectives has the form

$$\min_u F(u) \stackrel{\text{def}}{=} \ell_{N+1}(x_{N+1}) + \sum_{i=1}^{N} \ell_i(x_i, u_i) \tag{1}$$

$$x_{i+1} = f_i(x_i, u_i), \qquad i = 1, \cdots, N, \qquad x_1 = a \text{ (fixed)}; \tag{2}$$

where $x_i \in R^n$, $i = 1, \cdots, N+1$, and $u_i \in R^m$, $i = 1, \cdots, N$. The x_i variables are usually referred to as *states* and the u_i as *controls*. (The *costates* p_i are the Lagrange multipliers corresponding to the constraints (2).) In Dunn and Bertsekas [5] and Wright [14], algorithms which exhibit local quadratic convergence to nondegenerate minimizers of (1)–(2) are discussed. These algorithms are Newton's method (Algorithm III of [14]), and variants of sequential quadratic programming (Algorithms I and II of [14]), and they all require the solution of a linear-quadratic subproblem of the following form at each iteration:

$$\begin{aligned} \min_{v,y} \ & \textstyle\sum_{i=1}^{N} r_i^T v_i + z_i^T y_i + \frac{1}{2}(y_i^T Q_i y_i + 2 y_i^T R_i v_i + v_i^T S_i v_i) \\ & + z_{N+1}^T y_{N+1} + \tfrac{1}{2} y_{N+1}^T Q_{N+1} y_{N+1}, \\ & y_{i+1} = A_i y_i + B_i v_i + s_i, \qquad i = 1, \cdots, N, \qquad y_1 = s_0. \end{aligned} \tag{3}$$

Here, y_i, v_i, q_i denote the steps in x_i, u_i, p_i, respectively,

$$\begin{aligned} & Q_i = \frac{\partial^2 \mathcal{L}}{\partial x_i^2}; \qquad R_i = \frac{\partial^2 \mathcal{L}}{\partial x_i \partial u_i}; \qquad S_i = \frac{\partial^2 \mathcal{L}}{\partial u_i^2}; \\ & A_i = \frac{\partial f^i}{\partial x_i}; \qquad B_i = \frac{\partial f^i}{\partial u_i}; \qquad z_i = \frac{\partial \mathcal{L}}{\partial x_i}; \qquad r_i = \frac{\partial \mathcal{L}}{\partial u_i}; \\ & s_i = -x_{i+1} + f_i(x_i, u_i), \end{aligned} \tag{4}$$

* Received by the editors August 21, 1990; accepted for publication (in revised form) March 1, 1991. This research was supported by the Applied Mathematical Sciences subprogram of the Office of Energy research, United States Department of Energy, contract W-31-109-Eng-38, and by the National Science Foundation under contract DMS-8900984.

† Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois 60439.

and $\mathcal{L}$ is the Lagrangian for (1)–(2), specifically:

$$\mathcal{L}(u,x,p) = \ell_{N+1}(x_{N+1}) + \sum_{i=1}^{N} \ell_i(x_i,u_i) - \sum_{i=1}^{N} p_{i+1}^T(x_{i+1} - f_i(x_i,u_i)). \tag{5}$$

The first-order conditions for a set of vectors y_i, v_i, q_i to be a solution of (3) are the following system of linear equations:

$$r_i + R_i^T y_i + S_i v_i + B_i^T q_{i+1} = 0, \qquad i = 1,\cdots,N, \tag{6}$$

$$z_i + R_i v_i + Q_i y_i - q_i + A_i^T q_{i+1} = 0, \qquad i = 1,\cdots,N, \tag{7}$$

$$z_{N+1} + Q_{N+1} y_{N+1} - q_{N+1} = 0, \tag{8}$$

$$s_i - y_{i+1} + A_i y_i + B_i v_i = 0, \qquad i = 1,\cdots,N. \tag{9}$$

When the initial value $y_1 = s_0$ is included, the unknowns are ordered as

$$(q_1, y_1, v_1, q_2, y_2, v_2, \cdots, q_{N+1}, y_{N+1}),$$

and the equations are ordered appropriately, the coefficient matrix for this linear system becomes

$$\begin{bmatrix}
0 & -I & & & & & & & & & \\
-I & Q_1 & R_1 & A_1^T & & & & & & & \\
 & R_1^T & S_1 & B_1^T & & & & & & & \\
 & A_1 & B_1 & 0 & -I & & & & & & \\
 & & & -I & Q_2 & R_2 & A_2^T & & & & \\
 & & & & R_2^T & S_2 & B_2^T & & & & \\
 & & & & A_2 & B_2 & 0 & -I & & & \\
 & & & & & & -I & Q_3 & R_3 & A_3^T & \\
 & & & & & & & R_3^T & S_3 & B_3^T & \\
 & & & & & & & A_3 & B_3 & 0 & -I \\
 & & & & & & & & & -I & Q_4 \\
 & & & & & & & & & & \ddots \quad \ddots
\end{bmatrix} \tag{10}$$

with a right-hand side of

$$(-s_0^T, -z_1^T, -r_1^T, -s_1^T, -z_2^T, -r_2^T, -s_2^T, \cdots, -s_N^T, -z_{N+1}^T)^T. \tag{11}$$

The matrix is symmetric, with dimension $2n(N+1) + mN$, and a half-bandwidth of $(2n + m - 1)$.

A necessary and sufficient condition for (3) to have a unique, finite solution is that

$$\sum_{i=1}^{N} \frac{1}{2}(y_i^T Q_i y_i + 2 y_i^T R_i v_i + v_i^T S_i v_i) + \frac{1}{2} y_{N+1}^T Q_{N+1} y_{N+1} > 0 \tag{12}$$

for all y_i, v_i which satisfy

$$y_{i+1} = A_i y_i + B_i v_i, \qquad i = 1,\cdots,N, \qquad y_1 = 0. \tag{13}$$

Under this assumption, the point which satisfies the first-order conditions (6)–(9) is identical to the solution of (3). Second-order sufficient conditions for the vectors x_i^*,

u_i^*, p_i^* to be a solution of (1)–(2) include the requirement that (12)–(13) be satisfied when Q_i, R_i, etc., are defined by (4), evaluated at x_i^*, u_i^*, p_i^*. When the functions ℓ_i and f_i are sufficiently smooth, then (12)–(13) will also hold in some neighborhood of this optimal point. If (12)–(13) fails to hold, then the solution of (6)–(9) may be a saddle point for (3), and hence may not be a suitable search direction for the nonlinear algorithm. Clearly, modifications to the basic algorithms of [5] and [14] are needed to ensure that satisfactory global convergence properties will hold. Dunn and Bertsekas [5] suggest a Levenberg–Marquardt strategy, in which damping terms are added to the matrices S_i of (10). This does not destroy the structure of the matrix (10).

Another possible difficulty arises when the second derivatives of the functions f_i and ℓ_i are difficult to obtain. In this case, "pointwise" quasi-Newton methods, or finite differencing, can be used to replace the matrices Q_i, R_i, S_i by suitable approximations. (See Kelley and Sachs [8] for a pointwise quasi-Newton method for the infinite-dimensional analog of (1)–(2).) Again, the linear algebra task remains the same.

When pointwise constraints are applied to the controls u_i (for example, bounds on their magnitudes), projected gradient or active set methods give rise to linear systems similar to (10)–(11), in which transformation of the variable space leads to "reduction" of S_i, R_i, B_i, r_i. This is discussed further in §4.

From the above discussion, it should be clear that, implicitly or explicitly, the task of solving a problem of the form of (3) (or, alternatively, (10)–(11)) is at the heart of most practical algorithms for solving (1)–(2). Indeed, computationally speaking, it is usually the most time-consuming task. In this paper, we focus on this "inner loop," and present methods for efficiently solving (10)–(11) on multiprocessor computers. These methods take more account of the structure of the problem than the approach of [14], in which a general parallel bandsolver, with some modifications, was applied directly to (10)–(11). They are therefore faster, and lend themselves better to recursive, or "multilevel," implementation (see §3.3). The tradeoff is that, in contrast to the bandsolve approach, they may fail to find a solution to the system (10)–(11), even when the coefficient matrix is nonsingular. Such an occurrence is anticipated to be rare (it has not yet been observed on any test problems), but it would be wise to retain the bandsolve approach as a backup strategy.

The remainder of the paper proceeds as follows. In §2, known serial algorithms for efficiently solving (6)–(9) are described. Partitioning of the problem by what amounts to a domain decomposition, and corresponding modifications of the algorithms, are described in §3. In §4 we show how to modify the algorithms when constraints on the controls u_i are present, while in §5 some timing analysis of the performance of recursive implementations of these algorithms on multiprocessor architectures is given. Some results from an implementation on a shared-memory architecture are presented in §6. Finally, in §7, we discuss the continuous-time analog of (3), namely, the problem of finding functions $y : [0,T] \to R^n$ and $v : [0,T] \to R^m$ such that

$$\begin{aligned} (14)\quad & \int_0^T \tfrac{1}{2} y(t)^T Q(t) y(t) + y(t)^T R(t) v(t) + \tfrac{1}{2} v(t)^T S(t) v(t) + v(t)^T r(t) + y(t)^T z(t)\, dt \\ & \qquad + \tfrac{1}{2} y(T)^T Q_f Y(T) + y(T)^T z_f, \\ & \text{where} \quad \dot{y} = A(t) y + B(t) v + s(t), \qquad y(0) = y_0 \end{aligned}$$

is minimized. Here Q, R, S, r, z, A, B, and s are functions of appropriate dimensionality defined on the interval $[0,T]$. We show how the algorithms of §§2 and 3 can be adapted to this case.

A recent paper by Chang, Chang, and Luh [3] proposes a parallel algorithm for

(1)–(2) which has a similar flavor to the methods discussed here. They use an approach that is similar to multiple shooting for two-point boundary value problems, defining the state variables for certain equally spaced values of the index i as unknowns in a reduced optimization problem. The Hessian of this problem can be formed in parallel, and Newton steps are calculated by using cyclic reduction. The main differences from our algorithms are that, in [3], the problem is partitioned at the level of the nonlinear problem, and the algorithm consists of two distinct levels (whereas our methods can be implemented in up to $\lceil \log_2 N \rceil$ levels). We believe that exploiting parallelism at the level of the linear algebra allows more flexibility to enhance the algorithm at the nonlinear level (for example, by modifying it to handle constraints, to ensure global convergence, or to use approximate Hessians), without causing complications.

2. Dynamic programming. It is known that linear systems with dimension M and bandwidth K can be solved by using Gaussian elimination in $O(MK^2)$ operations. For the system (10)–(11), this translates to $O(N(m^3+n^3))$ operations. The dynamic programming approaches described below have just such a complexity bound; in fact, they can be thought of as specialized block elimination algorithms for this system.

One algorithm, described by Polak in [10], proceeds by first eliminating the v_i using equation (6). Substitution in (7)–(9) yields the system

$$\hat{Q}_i y_i + \hat{A}_i^T q_{i+1} - q_i + \hat{t}_i = 0, \qquad i = 1, \cdots, N, \tag{15}$$

$$-y_{i+1} + \hat{A}_i y_i + \hat{J}_i q_{i+1} + \hat{s}_i = 0, \qquad i = 1, \cdots, N, \tag{16}$$

where

$$\begin{aligned} \hat{A}_i &= A_i - B_i(S_i)^{-1}R_i^T, \\ \hat{J}_i &= -B_i(S_i)^{-1}B_i^T, \\ \hat{Q}_i &= Q_i - R_i(S_i)^{-1}R_i^T, \\ \hat{t}_i &= z_i - R_i(S_i)^{-1}r_i, \\ \hat{s}_i &= s_i - B_i(S_i)^{-1}r_i. \end{aligned}$$

Then, a Riccati substitution is made for q_i; that is, we seek matrices K_i and vector b_i such that

$$q_i = K_i y_i + b_i. \tag{17}$$

Clearly, from (8), $K_{N+1} = Q_{N+1}$ and $b_{N+1} = z_{N+1}$. By combining (15), (16), and (17), we can obtain expressions of the form

$$W_i y_i + w_i = 0, \qquad i = 1, \cdots, N, \tag{18}$$

where W_i depends on K_i and K_{i+1}, and w_i depends on b_i and b_{i+1}. Since (18) must be satisfied for all values of y_i, we deduce that $W_i = 0$ and $w_i = 0$. This yields recurrence relations for the K_i and b_i, which give rise to the following algorithm.

Algorithm RI

$K_{N+1} \leftarrow Q_{N+1}$, $b_{N+1} \leftarrow z_{N+1}$;

for $i = N, \cdots, 2$

$$\begin{aligned} K_i &\leftarrow \hat{A}_i^T[I - K_{i+1}\hat{J}_i]^{-1}K_{i+1}\hat{A}_i + \hat{Q}_i, \\ b_i &\leftarrow \hat{A}_i^T[I - K_{i+1}\hat{J}_i]^{-1}(K_{i+1}\hat{s}_i + b_{i+1}) + \hat{t}_i; \end{aligned}$$

$y_1 \leftarrow 0$;
for $i = 1, \cdots, N$

$$y_{i+1} \leftarrow (I - K_{i+1}\hat{J}_i)^{-T}[\hat{A}_i y_i + \hat{s}_i + \hat{J}_i b_{i+1}];$$

recover q_{i+1} from (17); recover v_i from (6).

Note that the factorizations of the matrices $(I - K_{i+1}\hat{J}_i)$ (needed in the first loop) can be re-used as factors of $(I - K_{i+1}\hat{J}_i)^T$ in the second loop.

In counting the higher-order terms in the operation count for this algorithm, we assume that advantage is taken of symmetry, where appropriate (K_i, $\hat{J}_i$, and $\hat{Q}_i$ are all symmetric), and that additions, multiplications, and divisions each count as one operation. We find that approximately

$$N(7n^3 + 4n^2 m + 4nm^2 + \tfrac{1}{3}m^3) \tag{19}$$

operations are needed for Algorithm RI.

We note for later reference that this algorithm can be trivially modified if a term involving q_{i+1} is introduced into equation (9). If (9) is replaced by

$$s_i - y_{i+1} + A_i y_i + B_i v_i + J_i q_{i+1} = 0, \tag{20}$$

we need only modify the definition of $\hat{J}_i$ to $J_i - B_i(S_i)^{-1}B_i^T$. The remainder of the algorithm is unchanged.

From the outset, RI depends on the nonsingularity of the matrices S_i. Although an analogous property is usually assumed to hold in the *continuous-time* problem (as we note in §7), Dunn and Bertsekas have noted that in the *discrete-time* case, this property may not hold even when (3) has a unique, finite solution. The algorithm described in [5] will, on the other hand, produce a solution for (3) whenever one exists. It proceeds by finding matrices θ_i and Γ_i, and vectors β_i and γ_i, such that

$$\begin{aligned} q_i &= \theta_i y_i + \beta_i, \qquad i = 1, \cdots, N+1 \\ v_i &= \Gamma_i y_i + \gamma_i, \qquad i = 1, \cdots, N. \end{aligned}$$

Substitution into the equations (6)–(9), and some manipulation, gives the following algorithm.

Algorithm DP

$\theta_{N+1} \leftarrow Q_{N+1}$; $\beta_{N+1} \leftarrow z_{N+1}$;

for $i = N, \cdots, 1$

$$\begin{aligned} \Gamma_i &\leftarrow -[S_i + B_i^T\theta_{i+1}B_i]^{-1}(R_i^T + B_i^T\theta_{i+1}A_i), \\ \gamma_i &\leftarrow -[S_i + B_i^T\theta_{i+1}B_i]^{-1}(r_i + B_i^T(\theta_{i+1}s_i + \beta_{i+1})), \\ \theta_i &\leftarrow (Q_i + A_i^T\theta_{i+1}A_i) + (R_i + A_i^T\theta_{i+1}B_i)\Gamma_i \qquad (i \neq 1), \\ \beta_i &\leftarrow A_i^T\theta_{i+1}(B_i\gamma_i + s_i) + A_i^T\beta_{i+1} + z_i + R_i\gamma_i \qquad (i \neq 1). \end{aligned}$$

The steps v_i and y_i can then be recovered in the following loop:

$y_1 = 0$;

for $i = 1, \cdots, N$
$v_i \leftarrow \Gamma_i y_i + \gamma_i$
$y_{i+1} \leftarrow A_i y_i + B_i v_i + s_i$.

The following operation count is obtained:

$$N(3n^3 + 5n^2 m + 3nm^2 + \tfrac{1}{3}m^3) + O(N(m^2 + n^2)). \tag{21}$$

As they stand, these algorithms can only exploit parallelism and vectorization *within* each iteration (i.e., in the matrix multiplications and factorizations), and hence reasonable efficiency can be expected only when m and n are fairly large. When the matrix (10) is sparse within its band, the situation is complicated further by a need for parallel sparse linear algebra algorithms. Moreover, the number of stages N is typically quite large. This is the motivation for considering parallelism *across the loop*, which we proceed to do in the next section.

3. Partitioned dynamic programming. Here we describe variants of the algorithms above which are more arithmetically expensive, but more amenable to parallel implementation. The problem is broken up into P partitions, where each adjacent pair of partitions is separated by a single stage. Within each partition, a variant of either DP or RI is used to express the "internal" variables for each partition in terms of the variables in the adjoining separator stages. A reduced problem consisting of P stages is then formed, and the process is repeated, possibly recursively. A detailed analysis of the possibilities appears in §5.

3.1. Partitioned version of DP. The initial partitioning is done by choosing "separator indices" $I_1, I_2, \cdots, I_P, I_{P+1}$ that lie in the range $1, 2, \cdots, N+1$ and satisfy the relationships

$$I_1 = 1; \qquad I_{j+1} \geq I_j + 2, \qquad (j = 1, \cdots, P); \qquad I_{P+1} = N + 1.$$

It follows from these requirements that P cannot exceed $(N+1)/2$. Usually, the separator indices are chosen to be spaced approximately equally, in which case they are approximately $(N+1)/P$ stages apart. The variables q_{I_j}, y_{I_j} and v_{I_j}, $j = 1, \cdots, P$ will be referred to as "separator variables." For these, it is convenient to use the notation

$$\tilde{q}_j \stackrel{\text{def}}{=} q_{I_j}; \qquad \tilde{y}_j \stackrel{\text{def}}{=} y_{I_j}; \qquad \tilde{v}_j \stackrel{\text{def}}{=} v_{I_j}. \tag{22}$$

These variables are the unknowns in the reduced system. Each partition consists of the indices strictly between two separators; partition j will consist of stages $I_j + 1$ to $I_{j+1} - 1$ inclusive.

Within each partition, we start by seeking matrices θ_i, Γ_i, D_i, and F_i and vectors β_i and γ_i such that

$$q_i = \theta_i y_i + D_i q_{I_{j+1}} + \beta_i, \qquad i = I_j + 1, \cdots, I_{j+1} - 1, \tag{23}$$

$$v_i = \Gamma_i y_i + F_i q_{I_{j+1}} + \gamma_i, \qquad i = I_j + 1, \cdots, I_{j+1} - 1. \tag{24}$$

Proceeding as in DP, using the equations (6)–(9) and (23)–(24), we can compute these matrices and vectors as follows.

Algorithm PDP

$$\theta_{I_{j+1}} \leftarrow 0; \qquad \beta_{I_{j+1}} \leftarrow 0; \qquad D_{I_{j+1}} \leftarrow I \tag{25}$$

for $i = I_{j+1} - 1, I_{j+1} - 2, \cdots, I_j + 1$

$$\Gamma_i \leftarrow -[S_i + B_i^T \theta_{i+1} B_i]^{-1}(R_i^T + B_i^T \theta_{i+1} A_i), \tag{26}$$

$$\gamma_i \leftarrow -[S_i + B_i^T \theta_{i+1} B_i]^{-1}(r_i + B_i^T(\theta_{i+1} s_i + \beta_{i+1})), \tag{27}$$

$$F_i \leftarrow -[S_i + B_i^T \theta_{i+1} B_i]^{-1} B_i^T D_{i+1}, \tag{28}$$

$$D_i \leftarrow (A_i + B_i \Gamma_i)^T D_{i+1}, \tag{29}$$

$$\theta_i \leftarrow (Q_i + A_i^T \theta_{i+1} A_i) + (R_i + A_i^T \theta_{i+1} B_i)\Gamma_i, \tag{30}$$

$$\beta_i \leftarrow A_i^T \theta_{i+1}(B_i \gamma_i + s_i) + A_i^T \beta_{i+1} + z_i + R_i \gamma_i. \tag{31}$$

We now aim to construct a reduced system in which only the separator variables are unknown. One equation can be deduced from (9) with $i = I_{j+1} - 1$, namely,

$$s_{I_{j+1}-1} + A_{I_{j+1}-1} y_{I_{j+1}-1} + B_{I_{j+1}-1} v_{I_{j+1}-1} = \tilde{y}_{j+1}.$$

The variables $v_{I_{j+1}-1}$ and $y_{I_{j+1}-1}$ can be eliminated by substituting from (24) with $i = I_{j+1} - 1$, and then (9) with $i = I_{j+1} - 2$. This substitution yields an equation in which the unknowns on the left-hand side are $y_{I_{j+1}-2}$ and $v_{I_{j+1}-2}$. This process is repeated for $i = I_{j+1} - 2$ down to $i = I_j$, at which point $\tilde{y}_j$ and $\tilde{v}_j$ appear. The resulting equation is

$$\tilde{A}_j \tilde{y}_j + \tilde{B}_j \tilde{v}_j + \tilde{J}_j \tilde{q}_{j+1} + \tilde{s}_j = \tilde{y}_{j+1}, \tag{32}$$

where

$$\begin{aligned}
\tilde{A}_j &= D_{I_j+1}^T A_{I_j}, \\
\tilde{B}_j &= D_{I_j+1}^T B_{I_j}, \\
\tilde{J}_j &= \sum_{i=I_j+1}^{I_{j+1}-1} D_{i+1}^T B_i F_i, \\
\tilde{s}_j &= \sum_{i=I_j+1}^{I_{j+1}-1} D_{i+1}^T (B_i \gamma_i + s_i) + D_{I_j+1}^T s_{I_j}.
\end{aligned}$$

A second equation is derived by setting $i = I_j$ in (7), $i = I_j + 1$ in (23), and $i = I_j$ in (9):

$$\tilde{t}_j + \tilde{R}_j \tilde{v}_j + \tilde{Q}_j \tilde{y}_j - \tilde{q}_j + \tilde{A}_j^T \tilde{q}_{j+1} = 0, \tag{33}$$

where

$$\begin{aligned}
\tilde{t}_j &= z_{I_j} + A_{I_j}^T(\theta_{I_j+1} s_{I_j} + \beta_{I_j+1}), \\
\tilde{R}_j &= R_{I_j} + A_{I_j}^T \theta_{I_j+1} B_{I_j}, \\
\tilde{Q}_j &= Q_{I_j} + A_{I_j}^T \theta_{I_j+1} A_{I_j}.
\end{aligned}$$

A third equation is derived by setting $i = I_j$ in (6), $i = I_j + 1$ in (23), and $i = I_j$ in (9):

$$\tilde{r}_j + \tilde{R}_j^T \tilde{y}_j + \tilde{S}_j \tilde{v}_j + \tilde{B}_j^T \tilde{q}_{j+1} = 0, \tag{34}$$

where

$$\tilde{r}_j = r_{I_j} + B_{I_j}^T(\theta_{I_j+1}s_{I_j} + \beta_{I_j+1}), \qquad \tilde{S}_j = S_{I_j} + B_{I_j}^T\theta_{I_j+1}B_{I_j},$$

and $\tilde{R}_j$ and $\tilde{B}_j$ are as defined above.

Two additional equations, for the first and last partitions, complete the system. From the initial condition, we have

$$\tilde{y}_1 = s_0, \tag{35}$$

while from (8), using $I_{P+1} = N + 1$, we have

$$\tilde{q}_{P+1} = Q_{N+1}\tilde{y}_{P+1} + z_{N+1}. \tag{36}$$

If the separator variables are ordered as

$$(\tilde{q}_1, \tilde{y}_1, \tilde{v}_1, \tilde{q}_2, \tilde{y}_2, \tilde{v}_2, \cdots, \tilde{q}_{P+1}, \tilde{y}_{P+1}),$$

and the right-hand sides are ordered as

$$(-s_0, -\tilde{t}_1, -\tilde{r}_1, -\tilde{s}_1, \cdots, -\tilde{t}_P, -\tilde{r}_P, -\tilde{s}_P, -z_{N+1}),$$

the coefficient matrix for (32)–(34), (35), and (36) has the following form:

$$\begin{bmatrix}
0 & -I & & & & & & & & & \\
-I & \tilde{Q}_1 & \tilde{R}_1 & \tilde{A}_1^T & & & & & & & \\
 & \tilde{R}_1^T & \tilde{S}_1 & \tilde{B}_1^T & & & & & & & \\
 & \tilde{A}_1 & \tilde{B}_1 & \tilde{J}_1 & -I & & & & & & \\
 & & & -I & \tilde{Q}_2 & \tilde{R}_2 & \tilde{A}_2^T & & & & \\
 & & & & \tilde{R}_2^T & \tilde{S}_2 & \tilde{B}_2^T & \ddots & & & \\
 & & & & & \ddots & \ddots & \ddots & & & \\
 & & & & & & & & \tilde{R}_P^T & \tilde{S}_P & \tilde{B}_P^T & \\
 & & & & & & & & \tilde{A}_P & \tilde{B}_P & \tilde{J}_P & -I \\
 & & & & & & & & & & -I & Q_{N+1}
\end{bmatrix}. \tag{37}$$

In comparison with DP, the main additional expenses in this partitioned algorithm involve calculation of the matrices F_i, D_i, $\tilde{J}_j$, $\tilde{A}_j$, $\tilde{B}_j$, $\tilde{Q}_j$, and $\tilde{R}_j$. Savings can be made in the first iteration of each loop (25)–(31) by taking note of the initial values of θ_{I_j} and D_{I_j}. Again, if lower-order terms are ignored, the operation count is approximately

$$(N - P)[6n^3 + 9n^2m + 5nm^2 + \tfrac{1}{3}m^3]. \tag{38}$$

All of the above constitutes phase I of the algorithm. The remaining phases consist of solving the reduced system (phase II) and recovering the internal states y_i from (9) and the internal controls and Lagrange multipliers v_i and q_i from (23) and (24) (phase III). The total operation count for phase III is $O(N(m^2 + n^2))$, so its cost is dominated by that of phase I unless m and n are very small. We do not take account of phase III in the timing analyses below. Phase II takes $O(P(m^3 + n^3))$ and, therefore, may be significant; this situation will be discussed further.

Unfortunately, in doing the partitioning of the dynamic programming algorithm, we lose the property that the inverses of the operators $[S_i + B_i^T \theta_{i+1} B_i]$ exist for all i. This phenomenon is similar to rank deficiency in the submatrices obtained by partitioning nonsingular banded linear systems (see, for example, [13]). The robustness of a parallel code could therefore be improved by including the algorithm of Wright [14] as a backup, in case PDP fails.

3.2. Partitioned version of RI. We now specify a partitioned version of RI applied to the equations (6), (7), (8), and (20). The controls can be eliminated as before, and the system (15)–(16) can be obtained, with the appropriate modification to the definition of $\hat{J}_i$. For i in the range I_j to $I_{j+1} - 1$, we make the following "Riccati" substitution for q_i:

$$q_i = K_i y_i + L_i^T \tilde{q}_{j+1} + b_i. \tag{39}$$

Manipulating the equations in the usual way, we obtain the following scheme for calculating K_i, L_i, and b_i.

Algorithm PRI (part 1)

$K_{I_{j+1}} \leftarrow 0$, $L_{I_{j+1}} \leftarrow I$, $b_{I_{j+1}} \leftarrow 0$;

for $i = I_{j+1} - 1, I_{j+1} - 2, \cdots, I_j$

$$\begin{aligned} K_i &\leftarrow \hat{A}_i^T [I - K_{i+1}\hat{J}_i]^{-1} K_{i+1} \hat{A}_i + \hat{Q}_i, \\ L_i^T &\leftarrow \hat{A}_i^T [I - K_{i+1}\hat{J}_i]^{-1} L_{i+1}^T, \\ b_i &\leftarrow \hat{A}_i^T [I - K_{i+1}\hat{J}_i]^{-1} (K_{i+1}\hat{s}_i + b_{i+1}) + \hat{t}_i. \end{aligned}$$

Again, we need to define a reduced system. One equation in this system can be found by setting $i = I_j$ in the formula (39), giving

$$\tilde{q}_j = K_{I_j} \tilde{y}_j + L_{I_j}^T \tilde{q}_{j+1} + b_{I_j}. \tag{40}$$

The second equation is derived by seeking matrices H_i and M_i, and vectors h_i, such that for $i = I_j, \cdots, I_{j+1} - 1$,

$$H_i y_i + M_i \tilde{q}_{j+1} - \tilde{y}_{j+1} + h_i = 0.$$

This can be achieved by the following loop.

Algorithm PRI (part 2)

$M_{I_{j+1}} \leftarrow 0$, $H_{I_{j+1}} \leftarrow I$, $h_{I_{j+1}} \leftarrow 0$.

for $i = I_{j+1} - 1, I_{j+1} - 2, \cdots, I_j$

$$\begin{aligned} H_i &\leftarrow H_{i+1}(I - \hat{J}_i K_{i+1})^{-1} \hat{A}_i, \\ M_i &\leftarrow M_{i+1} + H_{i+1}(I - \hat{J}_i K_{i+1})^{-1} \hat{J}_i L_{i+1}, \\ h_i &\leftarrow h_{i+1} + H_{i+1}(I - \hat{J}_i K_{i+1})^{-1} (\hat{J}_i b_{i+1} + \hat{s}_i). \end{aligned}$$

It is easy to see that $H_i = L_i$ for all i. Of course, parts 1 and 2 can be combined in the same loop. Setting $i = I_j$, we obtain the second equation of the reduced system:

$$L_{I_j} \tilde{y}_j + M_{I_j} \tilde{q}_{j+1} - \tilde{y}_{j+1} + h_{I_j} = 0. \tag{41}$$

It is important in this context to distinguish between the $K_{I_{j+1}}$ used as an initial value for the partition-j calculations (that is, $K_{I_{j+1}} = 0$), and the $K_{I_{j+1}}$ obtained as the end product of the partition-$(j+1)$ calculations. In constructing the reduced system, we assume the latter. Below, the notation $\tilde{K}_j$ is used for K_{I_j}, and so on, and we again use the additional equations (35)–(36) for the first and last partitions.

Ordering unknowns as

$$(\tilde{q}_1, \tilde{y}_1, \tilde{q}_2, \tilde{y}_2, \cdots, \tilde{q}_{P+1}, \tilde{y}_{P+1}),$$

and the right-hand side as

$$(-s_0, -\tilde{b}_1, -\tilde{h}_1, -\tilde{b}_2, \cdots, -\tilde{b}_P, -\tilde{h}_P, -z_{N+1}),$$

we get as the coefficient matrix for the reduced system

$$\begin{bmatrix} 0 & -I & & & & & \\ -I & \tilde{K}_1 & \tilde{L}_1^T & & & & \\ & \tilde{L}_1 & \tilde{M}_1 & -I & & & \\ & & \ddots & \ddots & & & \\ & & & & \tilde{L}_P & \tilde{M}_P & -I \\ & & & & & -I & Q_{N+1} \end{bmatrix}. \tag{42}$$

In counting operations for the algorithm so far, we note that S_i^{-1}, $\hat{J}_i$, $\hat{A}_i$, and $\hat{Q}_i$ must be calculated for all N stages. Savings can be made by noting that the first iteration of each of the loops above (that is, calculation of $K_{I_{j+1}-1}$, $L_{I_{j+1}-1}$, $M_{I_{j+1}-1}$) is virtually free; hence we multiply the cost for each iteration of these loops only by $(N-P)$. The higher-order terms sum to

$$N[15n^3 + 3n^2m + 5nm^2 + \tfrac{1}{3}m^3] - P[15n^3]. \tag{43}$$

3.3. Recursive implementation. The two partitioned algorithms above can be used in conjunction with the serial algorithms of §2 to devise recursive methods for solving (3). We note the following two facts:

- both the serial and partitioned dynamic programming algorithms have lower operation counts than the corresponding Riccati-based algorithms, but
- Algorithms RI and PRI can be used when (9) has been replaced by (20).

If we apply PDP to solve the original system (as we should, because of the first fact), we obtain a reduced system which contains equations of the form (20) instead of (9). Hence, because of the second fact, the reduced system, and any other reduced systems which arise at subsequent levels of recursion, can be solved only by using RI or PRI. In general, the optimal strategy for solving (3) consists of an application of PDP, followed by repeated applications of PRI (on fewer and fewer processors), followed by an application of RI on a single processor. Recall that we have the constraint that the size of the reduced system must decrease by a factor of at least 2 at each level of recursion; hence the theoretical limit on the number of levels is $\lceil \log_2 N \rceil$. In §5, we discuss "optimal" strategies for multiprocessor implementation.

The variant of the method in which the maximal number of levels is used is, effectively, a specialized form of cyclic reduction. It may be worth considering implementation of this variant on vector architectures, particularly when n and m are very small, so that factorizations involving individual blocks cannot be vectorized efficiently. We do not pursue this issue here, but refer the reader to Wright [15], in which cyclic-reduction–like algorithms for linear systems arising from general two-point boundary value problems are implemented and discussed.

4. Constrained problems. Usually, additional constraints are applied to the controls and/or states in the problem (1)–(2). The partitioned methods of the previous section can be extended in a straightforward way to the control-constrained case; this is the topic we discuss in this section.

Suppose that in (1)–(2) we have the additional constraints

$$g_i(u_i) \leq 0, \qquad i = 1, \cdots, N. \tag{44}$$

Often these constraints are simple bounds, or Cartesian products of simple geometric shapes such as spheres or cones (see Gawande and Dunn [6]). Algorithms such as two-metric gradient projection, or sequential quadratic programming with an active set strategy, may then be applied to solve (1), (2), and (44). It is not our aim to discuss the properties of these methods here, but rather to focus on the main computational tasks and their implementation. Both methods give rise to subproblems of the form (3), with the additional constraints

$$G_i v_i + g_i = 0, \tag{45}$$

where the vector g_i in (45) may contain only a subset of the components of $g_i(u_i)$ from (44). The terms in the objective function of (3) must be defined in terms of the modified Lagrangian

$$\mathcal{L}(u,x,p) = \ell_{N+1}(x_{N+1}) + \sum_{i=1}^{N} \ell_i(x_i,u_i) - \sum_{i=1}^{N} p_{i+1}^T(x_{i+1} - f_i(x_i,u_i)) + \sum_{i=1}^{N} \mu_i^T g_i(u_i).$$

First-order necessary conditions for (3), (45) yield the equations (7), (8), (9), (45), while (6) is replaced by

$$r_i + R_i^T y_i + S_i v_i + B_i^T q_{i+1} + G_i^T \mu_i = 0, \qquad i = 1, \cdots, N. \tag{46}$$

Standard null-space techniques can now be used to eliminate the μ_i and obtain a system of the form (6)–(9). Assuming without loss of generality that the G_i have full row rank, we can define orthogonal matrices U_i and upper triangular matrices T_i such that

$$U_i G_i^T = \begin{bmatrix} T_i \\ 0 \end{bmatrix}.$$

Partitioning U_i in the obvious way as

$$U_i = \begin{bmatrix} U_{i1} \\ U_{i2} \end{bmatrix},$$

and writing

$$\bar{v}_{i1} = U_{i1} v_i, \qquad \bar{v}_{i2} = U_{i2} v_i,$$

we obtain by substitution in (7), (9), (20), (45), and (46) the system

$$\bar{r}_{i2} + \bar{R}_{i2}^T y_i + \bar{S}_{i22} \bar{v}_{i2} + \bar{B}_{i2}^T q_{i+1} = 0, \qquad i = 1, \cdots, N, \tag{47}$$

$$\bar{t}_i + \bar{R}_{i2} \bar{v}_{i2} + Q_i y_i - q_i + A_i^T q_{i+1} = 0, \qquad i = 1, \cdots, N, \tag{48}$$

$$z_{N+1} + Q_{N+1} y_{N+1} - q_{N+1} = 0, \tag{49}$$

$$\bar{s}_i - y_{i+1} + A_i y_i + \bar{B}_{i2} \bar{v}_{i2} = 0, \qquad i = 1, \cdots, N. \tag{50}$$

Here $\bar{R}_{i2}$, $\bar{B}_{i2}$, $\bar{S}_{i22}$ are parts of the transformed matrices

$$\bar{R}_i = U_i R_i = \begin{bmatrix} \bar{R}_{i1} \\ \bar{R}_{i2} \end{bmatrix}, \quad \bar{B}_i = U_i B_i = \begin{bmatrix} \bar{B}_{i1} \\ \bar{B}_{i2} \end{bmatrix}, \quad \bar{S}_i = U_i S_i U_i^T = \begin{bmatrix} \bar{S}_{i11} & \bar{S}_{i12} \\ \bar{S}_{i21} & \bar{S}_{i22} \end{bmatrix},$$

while

$$\begin{aligned} \bar{r}_{i2} &= U_{i2} r_i + \bar{S}_{i21} \bar{v}_{i1}, \\ \bar{t}_i &= z_i + \bar{R}_{i1} \bar{v}_{i1}, \\ \bar{s}_i &= s_i + \bar{B}_{i1} \bar{v}_{i1}. \end{aligned}$$

The vectors $\bar{v}_{i1}$ and μ_i can be found from

$$\begin{aligned} \bar{v}_{i1} &= -T_i^{-T} g_i, \\ \mu_i &= -T_i^{-1} U_{i1} [r_i + R_i y_i + S_i \bar{v}_i + B_i^T q_{i+1}]. \end{aligned}$$

Clearly, the serial and parallel methods discussed in §§2 and 3 can be applied to solve (47)–(50), and this is typically the most expensive part of each major iteration. However, gradient projection algorithms for (1)–(2) perform two other significant calculations at each iteration. These are solution of the adjoint equation

$$\begin{aligned} p_{N+1} &= \nabla \ell_{N+1}(x_{N+1}), \\ p_i &= \frac{\partial f_i}{\partial x_i}^T p_{i+1} + \frac{\partial \ell_i}{\partial x_i}, \qquad i = N, \cdots, 1 \end{aligned} \tag{51}$$

(which is needed as part of the calculation of $\nabla F(u)$), and the evaluation of the nonlinear recurrence (2) for a given set of controls u_i. The technique for parallelizing (51) is essentially the same as that used in §3. The stages are partitioned into P groups, and within partition j, matrices E_i and vectors e_i are sought such that

$$p_i = E_i p_{I_{j+1}} + e_i, \qquad i = I_{j+1} - 1, \cdots, I_j + 1.$$

Recurrence relations for E_i and e_i can be derived.

When the state equations f_i are all linear, an identical technique can be used to speed up evaluation of (2). Here, in partition j, we seek H_i and h_i such that

$$x_i = H_i x_{I_j} + h_i, \qquad i = I_j + 1, \cdots, I_{j+1} - 1. \tag{52}$$

When the f_i are nonlinear, it is impossible to derive such linear relationships between the x_i which are the basis of the speedup techniques described in this paper. It is possible, in principle, to develop nonlinear analogs of (52), but implementation of these would require nonnumerical computing techniques which are outside the scope of this paper. One way around this bottleneck could be to use a two-level algorithm. At each outer iteration, the "IQP variant" of sequential quadratic programming is applied to (1), (2), and (44) to obtain a subproblem (3), with the additional linear inequality constraints

$$G_i v_i + g_i \leq 0, \qquad i = 1, \cdots, N.$$

This linearized problem (with linearized state equations) can then be solved by using two-metric gradient projection.

For problems with additional constraints on the states, quite different techniques from those above are required. This requirement is discussed in Psiaki and Park [11], who propose a recursive method involving variable elimination and pairwise combination of stages. We note that their method can be extended to allow merging of any number of successive stages at each level of recursion (not just 2), but refer the reader to [11] for details.

5. Optimal multiprocessor implementation. The operation count expressions derived in earlier sections can be used to give some insight into optimizing the number of levels of recursion, and optimizing the allocation of processors within each level for the algorithm of §3.3. The results of this section apply to the unconstrained problem, and to the cost of solving the reduced system (47)–(50) for the constrained problem. We start by introducing new notation for the operation counts of §§2 and 3. Since it is usually true that $m \leq n$, we write

$$m = \alpha n,$$

where $\alpha \in (0, 1]$. Then, scaling in each place by n^3, (19), (21), (38), and (43) can be replaced by

$$\begin{aligned} \text{Algorithm RI:} &\quad NA_\alpha, \\ \text{Algorithm DP:} &\quad NB_\alpha, \\ \text{Algorithm PRI:} &\quad NC_\alpha - PD_\alpha, \\ \text{Algorithm PDP:} &\quad (N-P)E_\alpha, \end{aligned}$$

where

$$\begin{aligned} A_\alpha &= 7 + 4\alpha + 4\alpha^2 + \tfrac{1}{3}\alpha^3, \\ B_\alpha &= 3 + 5\alpha + 3\alpha^2 + \tfrac{1}{3}\alpha^3, \\ C_\alpha &= 15 + 3\alpha + 5\alpha^2 + \tfrac{1}{3}\alpha^3, \\ D_\alpha &= 15, \\ E_\alpha &= 6 + 9\alpha + 5\alpha^2 + \tfrac{1}{3}\alpha^3. \end{aligned}$$

In the remainder of this section, quantities denoted by τ can be converted to absolute runtimes by multiplying by $n^3\delta$, where δ is the time required for one floating-point operation. For simplicity, however, we refer to the τ themselves as "runtimes."

5.1. Shared-memory machines. We start by discussing implementation on shared-memory multiprocessors. This is the simplest case, as we do not need to take account of interprocessor communication costs, so the operation counts above will give a reasonable indication of the elapsed time needed to solve a problem (3). However, when n is small, we also need to take into account the lower-order terms that were discarded earlier. When the total amount of data in the problem ($O(Nn^2)$) is large, the presence of hierarchical memory may also affect the times. This effect should at least be consistent as N increases, since in all algorithms, the data is processed in two or three sequential sweeps. Other costs which are ignored are costs of array indexing and manipulation of data structures, particularly the manipulation necessary to build up the reduced systems. The ratio of these costs to the arithmetic costs varies like $1/n$, so they may be significant when n is small.

On a shared-memory machine, the number of available processors P will typically be substantially smaller than the number of stages N. It seems reasonable, therefore, to consider a σ-level algorithm, in which PDP is first performed on all P processors, and then the reduced systems are solved by using PRI on fewer and fewer processors, until finally at the lowest level, RI is performed on a single processor. In general, at level k, a problem with P_k stages is solved on P_{k-1} processors ($P_\sigma = N$, $P_{\sigma-1} = P$, $P_0 = 1$). The runtime is proportional to

$$\tau_\sigma = \frac{N-P}{P} E_\alpha + \sum_{i=1}^{\sigma-2} \frac{P_{\sigma-i} C_\alpha - P_{\sigma-1-i} D_\alpha}{P_{\sigma-1-i}} + P_1 A_\alpha. \tag{53}$$

TABLE 1
Theoretical optimal processor allocations and scaled runtimes, where $P = 16$, $\alpha = .5$ *and* $N \geq 32$.

σ	P_1	P_2	P_3	P_4	P_5	$\tau_{\sigma,\text{opt}} - \tau_{DP}$
2	16.0					161
3	5.3	16.0				91.9
4	3.7	7.7	16.0			81.1
5	3.1	5.3	9.2	16.0		78.4
6	2.8	4.3	6.6	10.3	16.0	78.1

Minimizing τ_σ with respect to $P_{\sigma-2}, \cdots, P_1$, we find that the optimal values for the P_k are

$$P_{k,\text{opt}} = \left(\frac{C_\alpha}{A_\alpha}\right)^{(\sigma-1-k)/(\sigma-1)} P^{k/(\sigma-1)}, \qquad k = 1, \cdots, \sigma-2, \tag{54}$$

giving an optimal runtime of

$$\tau_{\sigma,\text{opt}} = \left(\frac{N}{P} - 1\right) E_\alpha + (\sigma-1) C_\alpha \left(\frac{A_\alpha P}{C_\alpha}\right)^{1/(\sigma-1)} - (\sigma-2) D_\alpha. \tag{55}$$

To illustrate this analysis, we insert some typical values of the parameters. Setting $\alpha = .5$ and $P = 16$, and assuming $N \geq 32$, values of $P_{k,\text{opt}}$ and $\tau_{\sigma,\text{opt}}$ are given in Table 1. We actually tabulate $\tau_{\sigma,\text{opt}} - \tau_{DP}$, where $\tau_{DP} = ((N/P) - 1)E_\alpha$, (that is, we exclude that part of the runtime expression which is independent of σ). The theoretical minimizing σ is 5.62, for which an adjusted runtime of 78.0 is obtained. Because we have the constraints that σ and the P_k are integers, that $P_k \geq 2P_{k-1}$, and that, preferably, P_k/P_{k-1} are integers, we use Table 1 as a guide and look for a nearby feasible schedule. The schedule $\sigma = 4$, $P_1 = 4$, $P_2 = 8$, $P_3 = 16$ produces a near-optimal adjusted runtime of 81.3. The five-stage schedule $P_1 = 2$, $P_2 = 4$, $P_3 = 8$, $P_4 = 16$ gives a time of 81.8. (We call this a "cyclic reduction" schedule, since after the initial execution of PDP on 16 processors, the size of the reduced system is halved at each level, as occurs in cyclic reduction applied to block-tridiagonal systems. See, for example, Golub and Van Loan [7].)

Finally we note that for systems with small numbers of processors, the scheduling above is in the nature of fine-tuning, since most of the work takes place at the top level in PDP. As mentioned, we have discounted the time required for this phase from the calculations in Table 1. In this example, $((N/P) - 1)E_\alpha$ would be 177 when $N = 256$ and 1498 when $N = 2048$.

5.2. Message-passing machines. We turn now to implementation on message-passing architectures. Here, we need to take into account the communication overhead needed in moving from one level of the algorithm to the next, and the availability of more processors than are typically found on shared-memory machines. Fortunately, the communication pattern is quite regular.

We assume that, at the top level of the algorithm, each of the P processors contains the data needed to execute its share of PDP. Processor j needs to have A_k, B_k, Q_k, R_k, S_k, s_k, z_k, and r_k for $k = I_j, \cdots, I_{j+1} - 1$. After doing its computation, this processor contributes the blocks $\tilde{A}_j$, $\tilde{B}_j$, $\tilde{J}_j$, $\tilde{Q}_j$, $\tilde{R}_j$, $\tilde{S}_j$, $\tilde{s}_j$, $\tilde{t}_j$, and $\tilde{r}_j$ to the reduced system. If processor j is not among the $P_{\sigma-2}$ processors which will be used

at the next level, it needs to send these blocks to another processor. The total length of the message will be $n^3 F_\alpha$, where $F_\alpha = (3 + 2\alpha + \alpha^2)/n + (2 + \alpha)/n^2$ words.

During the final phase, in which values of y_k, v_k, and q_k are being recovered, communication takes place in the reverse direction: processor j needs to know the values of $\tilde{q}_{j+1}$ and $\tilde{y}_j$. These may need to be sent from one of the $P_{\sigma-2}$ processors which were used at the next lower level. A total of $(P_{\sigma-1} - P_{\sigma-2})$ messages of length $n^3 G_\alpha$ words is involved, where $G_\alpha = 2/n^2$.

In general, communication between level $i + 1$ and level i requires a total of $P_i - P_{i-1}$ messages of length $n^3 F_\alpha$ to be sent (concurrently) during the first phase, and $P_i - P_{i-1}$ messages of length $n^3 G_\alpha$ to be sent (concurrently) during the second phase.

We assume that the time required to send a message of length M eight-byte words can be approximated by the formula

$$\gamma + \beta M, \tag{56}$$

where M is the number of words. This timing model has often been used for transfer of data between adjacent (i.e., directly connected) processors. However, newer hypercubes, such as the Intel iPSC/2 and the Ncube, use a routing mechanism which makes (56) a reasonable timing model for transfer between *any* two processors.

By modifying (53), we model the total runtime for a distributed-memory machine as

$$\tau_\sigma = \left(\frac{N}{P_{\sigma-1}} - 1\right) E_\alpha + \sum_{i=1}^{\sigma-2} \frac{P_{\sigma-i} C_\alpha}{P_{\sigma-1-i}} + P_1 A_\alpha - (\sigma - 2) D_\alpha + (\sigma - 1) H_\alpha, \tag{57}$$

where

$$H_\alpha = 2\frac{\gamma}{n^3 \delta} + (F_\alpha + G_\alpha)\frac{\beta}{\delta}.$$

Note that the new term is independent of $P_1, \cdots P_{\sigma-1}$ and so, for fixed σ, the optimal processor allocation will be the same as in the shared-memory case (54). Again we focus on the case in which the number of available processors P is less than $N/2$. For the optimal processor schedule (54), the runtime will be

$$\tau_{\sigma,\text{opt}} = \left(\frac{N}{P} - 1\right) E_\alpha + (\sigma - 1) C_\alpha \left(\frac{A_\alpha P}{C_\alpha}\right)^{1/(\sigma-1)} - (\sigma - 2) D_\alpha + (\sigma - 1) H_\alpha.$$

Again, we use some reasonable parameter values to obtain a feeling for performance. The experiments of Dunigan [4, Tables 3 and 6] show that, for double-precision arithmetic, approximate values of γ, β, and δ for the Ncube are

$$\gamma = 384\mu s, \qquad \beta = 20.8\mu s, \qquad \delta = 7.8\mu s.$$

We give predicted results for this machine, on an example in which $N = 2048$ and $\alpha = .5$. The number of available processors P is varied, as is the dimension of the state vector n. Most of the entries in Tables 2 and 3 are relative times, which are calculated by dividing each absolute time by the *time required for serial algorithm* DP, *executed on a single processor of the system*, and multiplied by 100 percent.

Table 2 gives the theoretical optimal values for σ and τ, together with the time τ_{DP} required for the top level of the process, namely, execution of algorithm PDP

TABLE 2
Ncube: optimal number of stages and normalized runtimes for $N = 2048$, $\alpha = .5$, and various P and n. Runtimes expressed as percentages of runtime for Algorithm DP *on a single processor.*

P	n	σ_{opt}	$\tau_{\text{opt}} - \tau_{DP}$	τ_{DP}
16	3	3.59	.83	11.63
	5	4.28	.71	
	10	4.91	.65	
64	3	5.22	1.28	2.84
	5	6.35	1.08	
	10	7.36	0.99	

TABLE 3
Ncube: best feasible schedules and normalized runtimes for $N = 2048$, $\alpha = .5$, and various P and n. Runtimes expressed as percentages of runtime for Algorithm DP *on a single processor.*

P	n	σ	schedule	$\tau_\sigma - \tau_{DP}$
16	3	4	4,8,16	.84
	5	4	4,8,16	.71
	10	5	2,4,8,16	.67
64	3	5	4,8,32,64	1.34
	5	6	4,8,16,32,64	1.09
	10	6	4,8,16,32,64	1.00

on the P available processors. Here, $\tau_{\text{opt}} = \tau_{\sigma,\text{opt}}$ with $\sigma = \sigma_{\text{opt}}$. Since the ratio of communication cost to computation cost is not too high for this machine, the σ_{opt} values are quite close to the cyclic reduction maximum of $\sigma = \log_2 P + 1$. Table 3 gives the best feasible schedules for the various P and n; it can be observed that the times required are very close to the theoretical optima from Table 2.

Efficiency of a P-processor algorithm can be defined by the general formula

$$\text{efficiency} = T_S/(P * T_P),$$

where T_S is the execution time for the best serial algorithm on a single processor, and T_P is the execution time for the P-processor parallel algorithm. From Table 2, we see, for example, that when $n = 10$ and $P = 16$, the parallel algorithm requires 12.3 percent of the execution time of the serial method. Hence, $T_P/T_S = .123$, and the efficiency is .51. When $n = 10$ and $P = 64$, the efficiency is still .41. These are quite competitive with the efficiencies attained in the shared-memory case.

A recently released hypercube, based on the Intel i860 chip, is characterized by a very high ratio of communication cost to computation cost. Despite this, a similar timing analysis on the problem above with the i860 parameters in place of the Ncube parameters showed that efficiencies of .42 and .21 could be still attained on 16 and 128 processors, respectively, for the problems in Tables 2 and 3. These results are very encouraging, given the fine-grained nature of the latter calculation.

6. Computational results. Results of the implementation of the algorithms on shared-memory and distributed-memory machines are given here. We use two simple test problems with small state and control dimension, but with a large number

of stages N. Both are discretizations of continuous-time problems, with an Euler discretization being applied to the original state equation

$$\dot{x}(t) = f(x, u, t).$$

PROBLEM 1 (Bertsekas [2]). ($m = 1$, $n = 2$.) Choosing $h = 1/N$,

$$\begin{aligned} &\min\ h\textstyle\sum_{i=1}^{N} 6u_i^2 + 2x_{i+1,1}^2 + x_{i+1,2}^2 \\ \text{s.t.}\quad &x_{i+1} = \begin{bmatrix} 1 & h \\ -h & 1 \end{bmatrix} x_i + \begin{bmatrix} 0 \\ h \end{bmatrix} u_i, \qquad i = 1, \cdots, N. \end{aligned}$$

PROBLEM 2 (Russell [12]). ($m = 2$, $n = 3$.) $h = 1/N$,

$$\begin{aligned} &\min\ h\textstyle\sum_{i=1}^{N} 4u_{i,1}^2 + u_{i,2}^2 + x_{N+1}^T Q x_{N+1}, \\ &\text{s.t.}\quad x_{i+1} = (I + h\hat{A})x_i + h\hat{B}u_i, \end{aligned}$$

where

$$Q = \begin{bmatrix} 5.2478 & -5.2896 & 0 \\ -5.2896 & 7.5183 & -1.6938 \\ 0 & -1.6938 & 1.3119 \end{bmatrix},$$

$$\hat{A} = \begin{bmatrix} -2 & 2 & 0 \\ -.25 & -2.1706 & 1.8752 \\ 0 & -1 & -1 \end{bmatrix}, \qquad \hat{B} = \begin{bmatrix} -.873 & 0 \\ 0 & -.873 \\ 0 & 0 \end{bmatrix}.$$

We also use both problems to test the constrained algorithms by imposing bounds on the components of u_i.

The Alliant FX/8, an eight-processor shared-memory machine, was used in the tests described below.

For the unconstrained problem, the following algorithms were implemented:

- Algorithm RI;
- Algorithm DP;
- The two-level algorithm consisting of PDP on eight processors, followed by RI to solve the reduced system;
- The three-level algorithm consisting of PDP on eight processors, followed by PRI on four processors and, finally, RI on a single processor.

The "serial" algorithms were compiled by using the `-Og` FORTRAN compiler option to run on a single processor, and also with the `-Ogc` option to run on all eight processors (and hence to reveal any parallelism in the algorithms). In the parallel version of RI, the initial elimination and final recovery of the state variables were done in a parallel loop. The two- and three-level parallel algorithms were also compiled to run both on a single processor and in concurrent mode. LINPACK routines were used to perform the various matrix factorizations and triangular solves.

Results are given in Table 4. Some improvement can be noted in the serial algorithms in concurrent mode, particularly for RI, because of the elementary optimization described above. The one-processor timings of the two- and three-level methods give an idea of how much "overhead" is involved in PDP, in the formation of matrices D_i, F_i, and $\tilde{J}_i$, and so on. It is seen that runtime increases by about 50 percent over serial DP. The eight-partition, eight-processor version of the two-level algorithms is about

TABLE 4
Alliant FX/8 runtimes (in seconds) for unconstrained Problems 1 and 2 ($N = 2000$).

	Problem 1		Problem 2	
	1 processor	8 processors	1 processor	8 processors
RI	3.15	1.90	6.56	3.25
DP	1.57	1.25	3.82	2.30
PDP—RI	2.48	.361	5.95	.820
PDP—PRI—RI	2.61	.379	6.27	.877

TABLE 5
Alliant FX/8 runtimes (in seconds) for gradient projection algorithm, Problem 1 ($N = 2000$).

	ubound=1.0 (18.4% active)		ubound=0.1 (81.5% active)	
	1 processor	8 processors	1 processor	8 processors
RI	11.8	7.94	17.8	10.1
DP	7.45	6.35	11.7	8.05
PDP—RI	12.2	2.18	14.2	2.76
PDP—PRI—RI	12.0	2.14	14.0	2.71

6.4 times faster than the eight-partition, one-processor version for the first problem and 7.1 times faster for the second problem. This speedup lags behind the ideal figure of 8, mainly because solution of the reduced system is a serial bottleneck. The three-level version of the algorithm took slightly longer, possibly because the overhead involved in additional levels of subroutine calls was not justified by the small amount of computation needed to solve the reduced system.

Defining "speedup" to be the ratio of the time taken by the best serial algorithm on one processor to the time taken for the best parallel algorithm on eight processors, we obtain a figure of 4.5 for the first problem and 4.6 for the second. These figures correspond well to the theoretical predictions of §5.

For the bound-constrained problem, the two-metric gradient projection framework from Bertsekas [2] is used, with each of the four unconstrained algorithms being used to solve the reduced system (47)–(50). The results are given in Tables 5 and 6. In the multiprocessor versions of the four codes, solution of the adjoint equation and evaluation of the states and objective function are parallelized as discussed in §4. The results are qualitatively similar to the unconstrained case, except that here the three-level algorithm has a slight advantage. Two factors inhibit perfect speedup in going from one to eight processors in the two- and three-level codes. The "serial" parts of these codes are proportionately more significant than in the unconstrained case, because of the small m and n values. There is also a load-balancing problem. An equal number of stages is assigned to each processor, but processing times for each partition vary because different numbers of u_i components are at their bounds within each partition. This means that the decrease factor in runtime in going from one to eight processors may be as low as 5.1 (as in Problem 1 with `ubound = 0.1`).

The number of iterations (i.e., solutions of reduced systems) is between 2 and 4 in each case. The sequence of iterates generated by each algorithm was the same, with the minor exception of the two serial methods, which because of roundoff error required an extra function evaluation of the last iteration of Problem 1 when the control bound was 0.1.

Speedups (ratio of best serial time to best parallel time) range from 3.5 to 4.3

TABLE 6
Alliant FX/8 runtimes (in seconds) for gradient projection algorithm, Problem 2 ($N = 2000$).

	ubound=0.2 (6.8% active)		ubound=0.05 (70.0% active)	
	1 processor	8 processors	1 processor	8 processors
RI	32.5	16.9	40.9	23.9
DP	21.4	14.3	27.8	17.8
PDP—RI	36.2	5.83	42.7	7.28
PDP—PRI—RI	35.3	5.63	42.0	7.13

and do not seem to depend strongly on the proportion of active constraints at the solution.

7. Continuous-time problems. The discrete-time problem and the algorithms discussed above have continuous-time analogs which we briefly describe in this section. These have an interesting relationship to known algorithms for two-point boundary value problems in ordinary differential and differential algebraic equations.

Given the problem (14), we can introduce the costate function q and apply standard constrained optimization techniques to obtain the following set of necessary conditions:

$$r(t) + R(t)^T y + S(t)v + B(t)^T q = 0, \tag{58}$$

$$\dot{q} + A(t)^T q + Q(t)y + R(t)v + z(t) = 0, \qquad q(T) = Q_f y(T) + z_f, \tag{59}$$

$$\dot{y} - A(t)y - B(t)v - s(t) = 0, \qquad y(0) = y_0. \tag{60}$$

This is a system of semi-explicit differential algebraic equations in q, v, and y; it will have index 1 if $S(t)$ is uniformly nonsingular on $[0,T]$. In fact, a standard second-order condition (the strengthened Legendre–Clebsch condition) for (14) to have a unique minimizer is that

$$S(t) \text{ is positive definite a.e. on } [0,T]. \tag{61}$$

It is, therefore, reasonable to use (58) to eliminate v from the system above and obtain the two-point boundary value problem

$$\dot{y} = \hat{A}(t)y + \hat{J}(t)q + \hat{s}(t), \qquad y(0) = y_0, \tag{62}$$

$$\dot{q} = -\hat{A}(t)^T q - \hat{Q}(t)y - \hat{z}(t), \qquad q(T) = Q_f y(T) + z_f, \tag{63}$$

where

$$\hat{A} = A - BS^{-1}R^T, \qquad \hat{J} = -BS^{-1}B^T, \qquad \hat{Q} = Q - RS^{-1}R^T,$$
$$\hat{s} = s - BS^{-1}r, \qquad \hat{z} = z - RS^{-1}r.$$

A continuous version of Algorithm RI can be deduced by making the Riccati substitution

$$q = K(t)y + b(t). \tag{64}$$

Combining (64) with (62)–(63), we obtain final value problems in K and b:

$$\dot{K} = -\hat{A}(t)^T K - K\hat{A}(t) - K\hat{J}(t)K - \hat{Q}(t), \qquad K(T) = Q_f, \tag{65}$$

$$\dot{b} = -[\hat{A}(t)^T - K(t)\hat{J}(t)]b - [K(t)\hat{s}(t) + \hat{z}(t)], \qquad b(T) = z_f. \tag{66}$$

Substitution into (62) then yields the following initial value problem for y:

$$\dot{y} = [\hat{A}(t) + \hat{J}(t)K(t)]y + [\hat{J}(t)b(t) + \hat{s}(t)], \qquad y(0) = y_0. \tag{67}$$

Hence it may be possible to solve (14) by integrating backward to solve (65) and (66), then integrating forward to solve (67), and finally obtaining q and v by substitution in (64) and (58), respectively. Of course, for this algorithm to work, we must be able to find a finite solution $K(t)$, $t \in [0,T]$, of (65). In the literature it is often simply assumed that such a solution exists (see, for example, Maurer [9]). However, Russell [12] and Polak [10] show that a finite $K(t)$ exists provided that, in addition to (61), we have

$$\begin{bmatrix} Q(t) & R(t) \\ R(t)^T & S(t) \end{bmatrix} \text{ is positive semidefinite a.e. on } [0,T]. \tag{68}$$

It is not difficult to derive an analog for Algorithm PRI. We first partition the interval $[0,T]$ into P subintervals using meshpoints $t_1, t_2, \cdots, t_{P+1}$ which satisfy

$$0 = t_1 < t_2 < \cdots < t_{P+1} = T.$$

Now we introduce the notation $\tilde{y}_i = y(t_i)$ and $\tilde{q}_i = q(t_i)$, $i = 1, \cdots, P+1$. The aim is now to express $y(t)$ and $q(t)$ purely in terms of $\tilde{y}_i$ and $\tilde{q}_i$, $i = 1, \cdots, P+1$, and to derive a "reduced" linear algebraic system in which the $\tilde{y}_i$ and $\tilde{q}_i$ are the unknowns. On each interval $[t_i, t_{i+1}]$ we make the "Riccati" substitution

$$q = K_i(t)y + b_i(t) + L_i(t)\tilde{q}_{i+1}, \qquad t \in [t_i, t_{i+1}]. \tag{69}$$

The usual manipulation yields final value problems for K_i, b_i, and L_i:

$$\dot{K}_i = -\hat{A}(t)^T K_i - K_i\hat{A}(t) - K_i\hat{J}(t)K_i - \hat{Q}(t), \qquad K_i(t_{i+1}) = 0, \tag{70}$$

$$\dot{b}_i = -[\hat{A}(t) + K_i(t)\hat{J}(t)]b_i - [K_i(t)\hat{s}(t) + \hat{z}(t)], \qquad b_i(t_{i+1}) = 0, \tag{71}$$

$$\dot{L}_i = -[\hat{A}(t)^T + K_i(t)\hat{J}(t)]L_i, \qquad L_i(t_{i+1}) = I. \tag{72}$$

By setting $\tilde{L}_i = L_i(t_i)$, $\tilde{K}_i = K_i(t_i)$, and $\tilde{b}_i = b_i(t_i)$, we deduce from (69) the following equation:

$$\tilde{q}_i = \tilde{K}_i\tilde{y}_i + \tilde{b}_i + \tilde{L}_i\tilde{q}_{i+1}.$$

To obtain another equation which relates the $\tilde{y}_i$ and $\tilde{q}_i$, we seek $H_i(t)$, $M_i(t)$, and $h_i(t)$ such that for $t \in [t_i, t_{i+1}]$,

$$H_i(t)y + M_i(t)\tilde{q}_{i+1} - \tilde{y}_{i+1} + h_i(t) = 0. \tag{73}$$

Substitution into (62) and (63) shows that

$$H_i(t) = L_i(t)^T, \tag{74}$$

and we obtain final-value problems in M_i and h_i:

$$\dot{M}_i = -L_i(t)^T\hat{J}(t)L_i(t), \qquad M_i(t_{i+1}) = 0, \tag{75}$$

$$\dot{h}_i = -L_i(t)^T[\hat{J}(t)b_i(t) + \hat{s}(t)], \qquad h_i(t_{i+1}) = 0. \tag{76}$$

By setting $t = t_i$ in (73) and using the notation $\tilde{M}_i = M_i(t_i)$, $\tilde{h}_i = h_i(t_i)$, we obtain

$$\tilde{L}_i^T\tilde{y}_i + \tilde{M}_i\tilde{q}_{i+1} - \tilde{y}_{i+1} + \tilde{h}_i = 0.$$

When the boundary conditions for $\tilde{y}_1$, $\tilde{q}_{P+1}$, and $\tilde{y}_{P+1}$ are included, we obtain a reduced linear system with unknown vector

$$(\tilde{q}_1, \tilde{y}_1, \tilde{q}_2, \tilde{y}_2, \cdots, \tilde{q}_{P+1}, \tilde{y}_{P+1}), \tag{77}$$

right-hand side

$$(-y_0^T, -\tilde{b}_1^T, -\tilde{h}_1^T, \cdots, -\tilde{h}_P^T, -z_f^T)^T, \tag{78}$$

and coefficient matrix

$$\begin{bmatrix} 0 & -I & & & & & & \\ -I & \tilde{K}_1 & \tilde{L}_1 & & & & & \\ & \tilde{L}_1^T & \tilde{M}_1 & -I & & & & \\ & & -I & \tilde{K}_2 & \tilde{L}_2 & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & & \tilde{L}_P^T & \tilde{M}_P & -I \\ & & & & & & -I & Q_f \end{bmatrix}. \tag{79}$$

The partitioned algorithm can be summarized as follows: we first obtain $\tilde{K}_i$, $\tilde{b}_i$, $\tilde{L}_i$, $\tilde{M}_i$, and $\tilde{h}_i$, for $i = 1, \cdots, P$ by integration of (70), (71), (72), (75), and (76), respectively. Then, (77)–(79) is solved to find $\tilde{q}_i$, $\tilde{y}_i$, $i = 1, \cdots, P+1$. Finally, $y(t)$ and $q(t)$ can be recovered by substituting into (73) and (69).

The scheme just described can be placed in the framework of "theoretical multiple shooting" applied to the boundary value problem (62)–(63) (see Ascher and Mattheij [1]). We are effectively solving a set of P subproblems, one on each interval $[t_i, t_{i+1}]$, and for the ith subproblem we enforce the separated end conditions

$$q(t_{i+1}) = \tilde{q}_{i+1}, \qquad y(t_i) = \tilde{y}_i. \tag{80}$$

(At this point, of course, $\tilde{q}_{i+1}$ and $\tilde{y}_i$ are unknowns.) This approach contrasts with "standard" multiple shooting in which only initial conditions are enforced for each subproblem. We are effectively using Riccati substitution techniques to find solutions to the particular and fundamental problems

$$\begin{aligned} \dot{y}_{p,i} &= \hat{A}(t) y_{p,i} + \hat{J}(t) q_{p,i} + \hat{s}(t), & y_{p,i}(t_i) &= 0, \\ \dot{q}_{p,i} &= -\hat{A}(t)^T q_{p,i} - \hat{Q}(t) y_{p,i} - \hat{z}(t), & q_{p,i}(t_{i+1}) &= 0, \end{aligned}$$

$$\begin{aligned} \dot{Y}_{1,i} &= \hat{A}(t) Y_{1,i} + \hat{J}(t) P_{1,i}, & Y_{1,i}(t_i) &= 0, \\ \dot{P}_{1,i} &= -\hat{A}(t)^T P_{1,i} - \hat{Q}(t) Y_{1,i}, & P_{1,i}(t_{i+1}) &= I, \end{aligned}$$

$$\begin{aligned} \dot{Y}_{2,i} &= \hat{A}(t) Y_{2,i} + \hat{J}(t) P_{2,i}, & Y_{2,i}(t_i) &= I, \\ \dot{P}_{2,i} &= -\hat{A}(t)^T P_{2,i} - \hat{Q}(t) Y_{2,i}, & P_{2,i}(t_{i+1}) &= 0. \end{aligned}$$

It can be shown that the following identities hold:

$$\begin{aligned} Y_{1,i}(t_{i+1}) &= \tilde{M}_i, & Y_{2,i}(t_{i+1}) &= \tilde{L}_i^T, & y_{p,i}(t_{i+1}) &= \tilde{h}_i, \\ P_{1,i}(t_i) &= \tilde{L}_i, & P_{2,i}(t_i) &= \tilde{K}_i, & q_{p,i}(t_i) &= \tilde{b}_i. \end{aligned}$$

Given this connection to multiple shooting, we can briefly address the remaining issues associated with the partitioned method. One is the existence of solutions of the subproblems on each interval. Again, the conditions (61), (68) restricted to the interval $[t_i, t_{i+1}]$ are sufficient for the existence of a finite solution K_i to (70). Then, under weak assumptions on the coefficient functions in (14), we can deduce existence of solutions to (71), (72), (75), and (76).

A second important issue is the stability of this procedure. It is easy to see from (72) and (75) that there is a possibility of exponential growth in L_i (and hence M_i). A large body of literature has appeared in recent years on stability of numerical methods for two-point boundary value problems; from this we can state, loosely speaking, that if (62)–(63) is well conditioned (that is, not too sensitive to perturbations in the boundary conditions), then the method outlined above will be stable provided that the interval lengths are sufficiently small, and the reduced system (77)–(79) is solved in a stable way. The system (77)–(79) has the same form as the system (15)–(16) arising from Algorithm RI, and hence could be solved by the discrete algorithms RI or PRI. If such a procedure turns out to be unstable (something which is easily detected substituting the calculated solution into (77)–(79) and finding the residual), a stable parallel solver along the lines of those discussed in Wright [15] can be used instead.

Finally, we note that the partitioned algorithm just outlined, with its special choice of boundary conditions on each interval, is by no means the only possible parallel algorithm for (62)–(63), though it does seem to be a reasonably intuitive one. Another possibility is to use a global finite-differencing scheme to set up a large block-banded matrix, and then to use parallelism at the level of the linear algebra (again, see [15] for details).

8. Conclusions. We have described parallel methods for optimal control problems, and described their implementation on various parallel architectures. In the discrete-time case, the parallelism is implemented at the computationally intensive "inner loop" in which a linear-quadratic regulator problem (3) must be solved. This allows flexibility in the choice of nonlinear optimization framework to be used to solve the problem (1)–(2). Good speedups over the best serial algorithms were observed on a shared-memory machine, and some simple timing analysis shows that good results can also be expected on distributed-memory architectures. Under appropriate assumptions, the continuous-time "limit" of the partitioned algorithms is a special multiple shooting algorithm.

Acknowledgments. I thank the referees for their perceptive comments, which improved the manuscript.

REFERENCES

[1] U. M. Ascher and R. M. M. Mattheij, *General framework, stability and error analysis for numerical stiff boundary value problems*, Numer. Math., 54 (1988), pp. 355–372.

[2] D. P. Bertsekas, *Projected Newton methods for optimization problems with simple constraints*, SIAM J. Control Optim., 20 (1982), pp. 221–246.

[3] S.-C. Chang, T.-S. Chang, and P. B. Luh, *A hierarchical decomposition for large-scale optimal control problems with parallel processing structure*, Automatica, 25 (1989), pp. 77–86.

[4] T. H. Dunigan, *Performance of the Intel i860 hypercube*, Tech. Report ORNL/TM-11491, Oak Ridge National Laboratory, Oak Ridge, TN, May 1990.

[5] J. C. DUNN AND D. P. BERTSEKAS, *Efficient dynamic programming implementations of Newton's method for unconstrained optimal control problems*, J. Optim. Theory Appl., 63 (1989), pp. 23–38.

[6] M. GAWANDE AND J. C. DUNN, *Variable metric gradient projection processes on convex feasible sets defined by nonlinear inequalities*, Appl. Math. Optim., 17 (1988), pp. 103–119.

[7] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, Second Edition, The Johns Hopkins University Press, Baltimore, MD, 1989.

[8] C. T. KELLEY AND E. W. SACHS, *A pointwise quasi-Newton method for unconstrained optimal control problems*, Numer. Math., 55 (1989), pp. 159–176.

[9] H. MAURER, *First and second order sufficient optimality conditions in mathematical programming and optimal control*, Math. Programming Stud., 14 (1981), pp. 163–177.

[10] E. POLAK, *Computational Methods in Optimization*, Academic Press, New York, 1970.

[11] M. L. PSIAKI AND K. PARK, *Trajectory optimization for real-time guidance: Part 1, Time-varying* LQR *on a parallel processor*, Proc. 1990 American Control Conference, May 23–25, 1990, San Diego, CA, pp. 248–253.

[12] D. L. RUSSELL, *Mathematics of Finite-Dimensional Control Systems*, Marcel Dekker, New York, 1979.

[13] S. J. WRIGHT, *Parallel algorithms for banded linear systems*, SIAM J. Sci. Statist. Comput., to appear; Preprint MCS–P64–0289, Argonne National Laboratory, Argonne, IL, 1989.

[14] ———, *Solution of discrete-time optimal control problems on parallel computers*, Parallel Comput., 16 (1990), pp. 221–238.

[15] ———, *Stable parallel algorithms for two-point boundary value problems*, SIAM J. Sci. Statist. Comput., to appear; Preprint MCS–P178–0990, Argonne National Laboratory, Argonne, IL, 1990.