Parallel Computing 16 (1990) 221–237
North-Holland

Solution of discrete-time optimal control problems on parallel computers *

Stephen J. WRIGHT

Department of Mathematics, Box 8205, North Carolina State University, Raleigh, NC27695, USA

Received August 1989
Revised January/June 1990

Abstract. We describe locally-convergent algorithms for discrete-time optimal control problems which are amenable to multiprocessor implementation. Parallelism is achieved both through concurrent evaluation of the component functions and their derivatives, and through the use of a parallel band solver which solves a linear system to find the step at each iteration. Results from an implementation on the Alliant FX/8 are described.

Keywords. Discrete-time optimal control problems, Constrained optimization problem, Parallel band solver, Multiprocessor implementation, Shared memory multiprocessor.

1. Introduction

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

$$\min_u F(u) \stackrel{\text{def}}{=} l_{N+1}(x^{N+1}) + \sum_{i=1}^{N} l_i(x^i, u^i) \tag{1.1}$$

$$\begin{aligned} x^1 &= a(\text{fixed}) \\ x^{i+1} &= f^i(x^i, u^i), \end{aligned} \tag{1.2}$$

where $x^i \in R^n$, $i = 1, \ldots, N+1$, and $u^i \in R^m$, $i = 1, \ldots, N$. The x^i variables are usually referred to as *states* and the u^i as *controls*. This problem may arise, for example as a discretization of the continuous-time problem

$$\min_{u(t)} \int_0^1 L(x(t), u(t), t)\, \mathrm{d}t + \phi(x(1))$$
$$\text{s.t.} \dot{x} = f(x, u, t), \quad x(0) = x_0,$$

(Polak [12], Jacobson and Mayne [7], Kelley and Sachs [8]), but it is also of interest in its own right in modelling certain physical processes.

In some situations there may be stagewise constraints on the x^i and u^i; in their most general form these can be written as

$$g^i(x^i, u^i) \leqslant 0, \quad i = 1, \ldots, N, \tag{1.3}$$

* Research partially performed while the author was visiting Argonne National Laboratory, Argonne, IL, and supported by the National Science Foundation under Grants DMS-8619903, DMS-8900984 and ASC-8714009, and the Air Force Office of Scientific Research under Grant AFOSR-ISSA-87-0092

where $g^i: R^n \times R^m \to R^{r_i}$. One important special case is the simple control-constrained problem in which we require

$$\| u^i \|_\infty \leqslant \eta, \quad i = 1, \ldots, N, \tag{1.4}$$

where η is some constant. In this paper, we consider the typical case in which N is much larger than m or n, and will describe the implementation of quadratically-convergent algorithms for solving these problems on shared-memory multiprocessors. It will be assumed throughout that

(i) the functions l_i, f^i and g^i are smooth enough (e.g. twice continuously differentiable) that the theoretical local convergence properties of these methods hold, and

(ii) the second derivatives of l_i, f^i and g^i with respect to x^i and u^i can be calculated.

A naive application of Newton's method to $F(u)$ would necessitate solving a dense linear system of order mN at each iteration – a task whose complexity is $O((mN)^3)$. However, algorithms have been known for some time for which the calculation of the step in u can be done in $O((m^3 + n^3)N)$ operations, and for which the local convergence is fast. Dunn and Bertsekas [4] have shown recently that the Newton step itself is obtainable in $O((m^3 + n^3)N)$ operations. In section 2.1, we motivate these Newton-like methods somewhat differently, by considering (1.1, 1.2) as a *constrained* optimization problem in which both the u^i and x^i vectors are unknowns. Three variants of the sequential quadratic programming method are presented, the last of which is in fact Newton's method applied to $F(u)$. The basis of all these methods is the solution of a large block-banded symmetric linear system, which can be done in $O((m^3 + n^3)N)$ operations, using the methods mentioned above. These are inherently serial, however, and one of the features of the parallel method which we propose for (1.1, 1.2) and (1.1, 1.2, 1.4) is the solution of this linear system using the parallel band solver from Wright [14]. The band solver can be tailored to take advantage of some of the special features of the linear system in question; this is discussed in Section 2.2. Some test problems from the literature and numerical results obtained on the Alliant FX/8 are given in Section 2.3. In Section 3.1, the solution of the general constrained problem (1.1, 1.2, 1.4) using active-set methods is outlined, while in Section 3.2, application of a gradient-projection-based approach to the simple control-constrained problem is discussed. For this latter application, two algorithms (one from Dunn [3] and one which uses the parallel band solver) are described; some numerical results are given in Section 3.3.

Superscripts are used to distinguish between vector and matrix quantities at different stages $i = 1, 2, \ldots, N+1$. Subscripts on scalars serve this same purpose. Subscripts on vectors and matrices indicate iteration number.

In keeping with the usual conventions, we say that a sequence $\{a_k\}$ is R-quadratically convergent to a_* if there exists a sequence of scalars $\{\eta_k\}$ such that

$$\| a_k - a_* \| \leqslant \eta_k \tag{1.5}$$

and

$$\limsup \frac{\eta_{k+1}}{\eta_k^2} < \infty.$$

We say that $\{a_k\}$ is Q-quadratically convergent if the inequality in (1.5) can be replaced by an equality.

To avoid cluttering of the expressions in Section 2, we omit the transpose superscript 'T' on vectors where this does not cause confusion. For example the expression $y^T Q y$ will be written as yQy, and $y^T x$ will be yx. (The superscript is retained for matrix transposes.) Another notational convenience in the remainder of the paper is to use u, x, and p defined by

$$u^T = (u^{1T}, u^{2T}, \ldots, u^{NT})$$
$$x^T = (x^{2T}, x^{3T}, \ldots, x^{N+1T})$$
$$p^T = (p^{2T}, p^{3T}, \ldots, p^{N+1T}).$$

(p^{i+1} is the Lagrange multiplier vector for the 'constraint' (1.2).) In keeping with omission of the transpose on vectors, (u, x, p) will be used for $\begin{pmatrix} u \\ x \\ p \end{pmatrix}$.

2. The unconstrained problem

2.1 Algorithms with fast local convergence

Specialized algorithms for solving (1.1) which are both quadratically convergent and for which the cost of calculating successive iterates increases only linearly with N have been known for some time. The best-known examples are the differential dynamic programming (DDP) algorithm of Jacobson and Mayne [7] and the method described by Polak [12, pp. 92–102]. The DDP algorithm is quadratically convergent in u, since it produces steps which are asymptotically similar to Newton steps. It was used in preference to Newton's method because it was apparently believed for some time that the Newton step is much more expensive to calculate. This is not true; Dunn and Bertsekas [4] show that the Newton step can be obtained by using dynamic programming to solve a linear-quadratic optimal control problem. Polak's approach also solves a linear-quadratic subproblem.

In this section, we consider the treatment of (1.1, 1.2) as a problem in constrained optimization, where x and u are both variable and the state equations (1.2) are constraints. Application of the sequential quadratic programming (SQP) method yields, under the usual second-order sufficiency assumptions, Q-quadratic convergence in the triple (u, x, p), where the vector p is made up of Lagrange multipliers p^{i+1} which correspond to the 'constraint' $x^{i+1} = f^i(x^i, u^i)$. Modifications to the SQP algorithm produce variants which are Q-quadratically convergent in (u, x) and in u alone; the latter is in fact Newton's method. These variants are successively less parallelizable, and since there is little practical difference in the convergence rates (all are R-quadratically convergent in u), SQP should certainly be considered as an alternative to Newton's method, at least in the neighbourhood of a solution. We take up this point later.

We assume throughout that there are vectors u_* and x_* which satisfy second-order sufficient conditions to be a solution of (1.1). These are defined after we have introduced some notation. Further, we assume that all iterates are sufficiently close to u_* and x_* that the usual local convergence properties hold. Convergence to a solution from an arbitrary starting point is of course desirable from a practical point of view – a Levenberg-style modification is discussed in Dunn and Bertsekas [4] which would help in this aim. We put this issue aside here, since we are mainly concerned with the basic algorithm and its implementation in a parallel setting.

The Lagrangian for (1.1) is, using the notation already introduced for Lagrange multipliers,

$$\mathscr{L}(u, x, p) = l_{N+1}(x^{N+1}) + \sum_{i=1}^{N} l_i(x^i, u^i) - \sum_{i=1}^{N} \left(x^{i+1} - f^i(x^i, u^i)\right) p^{i+1}. \tag{2.1}$$

First-order conditions for (u, x, p) to be a solution of (1.1) are that

$$\frac{\partial \mathscr{L}}{\partial u^i} = \frac{\partial l_i}{\partial u^i} + \frac{\partial f^i}{\partial u^i} p^{i+1} = 0, \quad i = 1, \ldots, N \tag{2.2}$$

$$\frac{\partial \mathscr{L}}{\partial x^i} = \frac{\partial l_i}{\partial x^i} - p^i + \frac{\partial f^i}{\partial x^i} p^{i+1} = 0, \quad i = 2, \ldots, N \tag{2.3}$$

$$\frac{\partial \mathscr{L}}{\partial x^{N+1}} = \frac{\partial l_{N+1}}{\partial x^{N+1}} - p^{N+1} = 0 \tag{2.4}$$

$$\frac{\partial \mathscr{L}}{\partial p^{i+1}} = -x^{i+1} + f^i(x^i, u^i) = 0, \quad i = 1, \ldots, N. \tag{2.5}$$

(Here we have fixed $x^1 = a$ and eliminated it from the problem.) We introduce the following notation, some of which has been used elsewhere (Dunn [3], Dunn and Bertsekas [4]), although defined there in terms of Hamiltonians rather than the Lagrangian.

$$Q^i = \frac{\partial^2 \mathscr{L}}{\partial (x^i)^2}; \qquad R^i = \frac{\partial^2 \mathscr{L}}{\partial x^i \, \partial u^i}; \qquad S^i = \frac{\partial^2 \mathscr{L}}{\partial (u^i)^2};$$
$$A^i = \frac{\partial f^i}{\partial x^i}; \qquad B^i = \frac{\partial f^i}{\partial u^i}; \qquad t^i = \frac{\partial \mathscr{L}}{\partial x^i}; \qquad r^i = \frac{\partial \mathscr{L}}{\partial u^i}. \tag{2.6}$$

Assumption 1. There exist vectors u_* and x_* which satisfy second-order sufficient conditions for (1.1), that is,

(i) there exists at least one vector p_* such that the equations (2.2–2.5) hold (with all quantities evaluated at (u_*, x_*)),

(ii) for some p_* satisfying (i), and all nonzero directions (v, y) such that

$$y^1 = 0$$
$$y^{i+1} = A^i_* y^i + B^i_* v^i$$

we have that

$$(v, y) \nabla^2 \mathscr{L}(u_*, x_*, p_*)(v, y) > 0,$$

that is,

$$\tfrac{1}{2} y^{N+1} Q^{N+1}_* y^{N+1} + \tfrac{1}{2} \sum_{i=1}^{N} \left(y^i Q^i_* y^i + 2 y^i R^i_* v^i + v S^i_* v^i \right) > 0$$

(where the subscript '$_*$' denotes evaluation at (u_*, x_*, p_*)).

It is well known (see Fletcher [5]) that in the neighbourhood of a point (u_*, x_*) satisfying Assumption 1, SQP is equivalent to applying Newton's method to (2.2–2.5). In the following, we use v^i, y^i and q^i to denote components of the Newton steps in the control, state and multiplier vectors (δu, δx and δp) respectively. Linearization of (2.2–2.5) gives the following system to be solved to find these quantities:

$$r^i + R^{i\mathrm{T}} y^i + S^i v^i + B^{i\mathrm{T}} q^{i+1} = 0, \qquad i = 1, \ldots, N, \tag{2.7}$$
$$t^i + R^i v^i + Q^i y^i - q^i + A^{i\mathrm{T}} q^{i+1} = 0, \qquad i = 1, \ldots, N, \tag{2.8}$$
$$t^{N+1} + Q^{N+1} y^{N+1} - q^{N+1} = 0, \tag{2.9}$$
$$\left[-x^{i+1} + f^i(x^i, u^i) \right] - y^{i+1} + A^i y^i + B^i v^i = 0, \qquad i = 1, \ldots, N. \tag{2.10}$$

Using the notation

$$\hat{r}^i = \frac{\partial l_i}{\partial u^i}, \quad \hat{t}^i = \frac{\partial l_i}{\partial x^i}, \quad \hat{s}^i = -x^{i+1} + f^i(x^i, u^i), \quad p_+ = p + q,$$

(2.7–2.10) can be written alternatively as

$$\hat{r}^i + R^{i\mathrm{T}} y^i + S^i v^i + B^{i\mathrm{T}} p^{i+1}_+ = 0, \qquad i = 1, \ldots, N, \tag{2.11}$$
$$\hat{t}^i + R^i v^i + Q^i y^i - p^i_+ + A^{i\mathrm{T}} p^{i+1}_+ = 0, \qquad i = 2, \ldots, N, \tag{2.12}$$
$$\hat{t}^{N+1} + Q^{N+1} y^{N+1} - p^{N+1}_+ = 0, \tag{2.13}$$
$$\hat{s}^i - y^{i+1} + A^i y^i + B^i v^i = 0, \qquad i = 1, \ldots, N. \tag{2.14}$$

The (locally equivalent) SQP form of (2.7–2.10) is of course

$$\min_{\substack{v,y \\ y^1=0}} \sum_{i=1}^{N} r^i v^i + t^i y^i + \tfrac{1}{2}\left(y^i Q^i y^i + 2 y^i R^i v^i + v^i S^i v^i \right) + t^{N+1} y^{N+1} + \tfrac{1}{2} y^{N+1} Q^{N+1} y^{N+1} \tag{2.15}$$

$$\text{s.t. } y^{i+1} = A^i y^i + B^i v^i + \hat{s}^i, \quad i = 1, \ldots, N.$$

where the q^i are the Lagrange multipliers for the (linear) constraints at the solution of (2.15). The system (2.11–2.14) has a similar quadratic-programming representation, with the optimal Lagrange multipliers being p_+^{i+1} instead of q^{i+1}.

We can summarize the resulting algorithm as follows:

Algorithm I (SQP/Lagrange-Newton).

$k \leftarrow 0$

repeat

1. Solve (2.7–2.10) (or the alternatives) for v^i, y^i and q^{i+1} (or p_+^{i+1})
2. Set

$$\begin{aligned} u^i &\leftarrow u^i + v^i, && i = 1, \ldots, N \\ x^i &\leftarrow x^i + y^i, && i = 2, \ldots, N+1 \\ p^{i+1} &\leftarrow p^{i+1} + q^{i+1} = p_+^{i+1}, && i = 1, \ldots, N \end{aligned}$$

3. $k \leftarrow k + 1$

until convergence.

The following convergence result is a standard one for SQP methods; its proof is omitted.

Theorem 2.1. *Let u_* and x_* satisfy the second-order sufficient conditions for* (1.1) *with optimal Lagrange multipliers p_*. Denote the sequence of iterates produced by Algorithm I as (u_k, x_k, p_k). Then if the initial estimates (u_0, x_0, p_0) are sufficiently close to (u_*, x_*, p_*), the estimate*

$$\| (u_{k+1} - u_*, x_{k+1} - x_*, p_{k+1} - p_*) \| = \mathrm{O}\left(\| (u_k - u_*, x_k - x_*, p_k - p_*) \|^2 \right)$$

holds, that is, the sequence $\{(u_k, x_k, p_k)\}$ is Q-quadratically convergent.

One variant of this approach is to update the multipliers p^{i+1} using the formulae (2.3–2.4), rather than setting $p^{i+1} \leftarrow p_+^{i+1}$. The resulting algorithm is

Algorithm II.

$k \leftarrow 0$

repeat

1. Solve (2.7–2.10) (or the alternatives) for v^i, y^i and q^{i+1} (or p_+^{i+1})
2. Set

$$\begin{aligned} u^i &\leftarrow u^i + v^i, && i = 1, \ldots, N \\ x^i &\leftarrow x^i + y^i, && i = 2, \ldots, N+1 \end{aligned}$$

3. Evaluate A^i, t^i, $i = 2, \ldots, N$ (at the *new* (u, x) iterate)
4. $p^{N+1} \leftarrow t^{N+1}$; **for** $i = N, \ldots, 2$ $\quad p^i \leftarrow t^i + A^{i\mathrm{T}} p^{i+1}$;
5. $k \leftarrow k + 1$

until convergence.

Step 4 is the solution of the adjoint equation, an essential component of the methods in [4,12]. No extra function evaluations are needed here, since A^i and t^i can of course be re-used in Step 1 of the following iteration. However, Step 4 does seem to introduce an inherently sequential element into the algorithm (although some speedup of this step is possible in a parallel environment, using techniques discussed in Section 2.2). The convergence of Algorithm II is slightly different from that of Algorithm I. Denoting the updated value of (u, x) obtained in Step 3 of iteration k as (u_k, x_k), and the corresponding p obtained in Step 4 by p_k, it is easy to show using an inductive argument based on (2.3–2.4) that

$$\| p_k - p_* \| = O(\| (u_k - u_*, x_k - x_*) \|).$$

This observation leads directly to the following result:

Theorem 2.2. *Let u_*, x_* and p_* be as in Theorem* 2.1, *and suppose that the starting point (u_0, x_0) is sufficiently close to (u_*, x_*) and that the initial p_0 is obtained by performing Steps* 3 *and* 4 *of Algorithm II at (u_0, x_0). Then the sequence of iterates generated by Algorithm II satisfies*

$$\| (u_{k+1} - u_*, x_{k+1} - x_*) \| = O(\| (u_k - u_*, x_k - x_*) \|^2).$$

Proof.

$$\begin{aligned}
&\| (u_{k+1} - u_*, x_{k+1} - x_*) \| \\
&\leqslant \| (u_{k+1} - u_*, x_{k+1} - x_*, p_k + q_k - p_*) \| \\
&= O(\| (u_k - u_*, x_k - x_*, p_k - p_*) \|^2) \text{ by Theorem 1} \\
&= O(\| (u_k - u_*, x_k - x_*) \|^2) \text{ by the observation above.} \quad \square
\end{aligned}$$

A second variant is obtained by evaluating x^i, $i = 2, \ldots, N+1$ from the original state equations (2.5), rather than by setting $x^i \leftarrow x^i + y^i$:

Algorithm III.

$k \leftarrow 0$

repeat

1. Solve (2.7–2.10) (or the alternatives) for v^i, y^i and q^{i+1} (or p_+^{i+1})
2. Set
$$u^i \leftarrow u^i + v^i, \quad i = 1, \ldots, N$$
3. $x^1 \leftarrow a$; **for** $i = 1, \ldots, N$ $\quad x^{i+1} \leftarrow f^i(x^i, u^i)$;
4. Evaluate A^i, t^i, $i = 2, \ldots, N$ (at the *new* (u, x) iterate)
5. $p^{N+1} \leftarrow t^{N+1}$; **for** $i = N, \ldots, 2$ $\quad p^i \leftarrow t^i + A^{iT} p^{i+1}$;
6. $k \leftarrow k + 1$

until convergence.

Clearly, Step 3 is inherently serial, since each x^{i+1} may depend nonlinearly on x^i. The linear-quadratic subproblem to be solved in Step 1 simplifies a little because of the choice of x and p; now $\hat{s}^i = 0$, $i = 1, \ldots, N$, and $t^i = 0$, $i = 1, \ldots, N+1$. Making these substitutions in (2.15) and comparing with Dunn and Bertsekas [4, equation (19)], we see that Algorithm III is simply Newton's method applied to $F(u)$! This is of course known to be quadratically convergent in u. We can also see this directly by using an inductive argument to show that x_k obtained in Step 3 has

$$\| x_k - x_* \| = O(\| u_k - u_* \|),$$

and also that p_k obtained in Steps 4 and 5 has

$$\| p_k - p_* \| = O(\| (u_k - u_*, x_k - x_*) \|) = O(\| u_k - u_* \|).$$

Application of Theorem 2.2 then yields the following result, which we state for completeness:

Theorem 2.3. *Let u_*, x_*, p_* be as in Theorem* 2.1, *and suppose that the starting vector u_0 is sufficiently close to u_* and that the initial x_0 and p_0 are obtained by executing Steps* 3, 4 *and* 5, *with $u = u_0$. Then the sequence of iterates generated by Algorithm III satisfies*

$$\| u_{k+1} - u_* \| = O(\| u_k - u_* \|^2).$$

In passing, we note from [4, equations (10b) and (18a)] that the condition (*ii*) in Assumption 1 is equivalent to positive definiteness of the Hessian $\nabla^2 F(u^*)$, and hence second-order sufficient conditions for (1.1) (regarded as a constrained problem) are equivalent to sufficient conditions for u_* to be a strict local minimizer of $F(u)$.

In all these algorithms, Step 1 typically uses most of the computational effort, since it does the evaluation of most of the functions and derivatives, and also solves a large linear system to find the various components of the step. The evaluations are independent of each other and can be carried out concurrently. Solution of the linear system is more complicated. We describe the options available for this part of the problem with reference to the formulae (2.7–2.10), although, as has been noted, a few terms disappear in Algorithms II and III.

The simplest option is to assemble the equations into a large, symmetric (but indefinite), block-banded linear system, and solve it directly. On shared-memory machines, the parallel band solver discussed in [14] can be used. We explore this possibility in Section 2.2, and show in Section 2.3 that this option is the fastest in many situations.

Polak's recurrence [12] proceeds by assuming nonsingularity of each S^i, $i = 1, \ldots, N$. Equation (2.7) is used to eliminate v^i from the problem. The following algorithm is then used to generate matrices K^i and vectors b^i ($i = 2, \ldots, N+1$) such that

$$q^i = K^i y^i + b^i. \tag{2.16}$$

(We omit details of the derivation.)

Algorithm P.

$$K^{N+1} \leftarrow Q^{N+1}, \; b^{N+1} \leftarrow t^{N+1};$$

for $i = N, \ldots, 2$

$$K^i \leftarrow \tilde{A}^{i\mathrm{T}} [I - K^{i+1} \tilde{J}^i]^{-1} K^{i+1} \tilde{A}^i + \tilde{Q}^i$$

$$b^i \leftarrow \tilde{A}^{i\mathrm{T}} [I - K^{i+1} \tilde{J}^i]^{-1} (K^{i+1} \tilde{f}^i + b^{i+1}) + \tilde{t}^i$$

where

$$\tilde{A}^i = A^i - B^i T^i R^{i\mathrm{T}}$$

$$\tilde{J}^i = -B^i T^i B^{i\mathrm{T}}$$

$$\tilde{Q}^i = Q^i - R^i T^i R^{i\mathrm{T}}$$

$$\tilde{t}^i = t^i - R^i T^i r^i$$

$$\tilde{f}^i = \hat{s}^i - B^i T^i r^i$$

and

$$T^i = (S^i)^{-1}.$$

Further manipulation yields a recurrence involving the δx components alone:

$$y^{i+1} = (I - K^{i+1}\tilde{J}^i)^{-\mathrm{T}}\left[\tilde{A}^i y^i + \tilde{f}^i + \tilde{J}^i b^{i+1}\right],$$

which, given $y^1 = 0$, is sufficient to determine δx completely. (2.16) can then be used to recover q^i, and v can be found from (2.7).

The recurrence described in Dunn and Bertsekas [4] does not rely on nonsingularity of the S^i, and is in fact guaranteed to successfully find the Newton step in a neighbourhood of the minimizer, where the Hessian $\nabla^2 F(u)$ is positive definite. It proceeds by finding matrices Θ^i and Γ^i, and vectors β^i and γ^i, such that

$$q^i = \Theta^i y^i + \beta^i, \quad i = 2, \ldots, N+1$$
$$v^i = \Gamma^i y^i + \gamma^i, \quad i = 1, \ldots, N.$$

Manipulation of (2.7–2.10) yields the following algorithm:

Algorithm DB.

$$\Theta^{N+1} \leftarrow Q^{N+1}; \ \beta^{N+1} \leftarrow t^{N+1};$$

for $i = N, \ldots, 1$

$$\Gamma^i \leftarrow -\left[S^i + B^{i\mathrm{T}}\Theta^{i+1}B^i\right]^{-1}\left(R^{i\mathrm{T}} + B^{i\mathrm{T}}\Theta^{i+1}A^i\right)$$
$$\gamma^i \leftarrow -\left[S^i + B^{i\mathrm{T}}\Theta^{i+1}B^i\right]^{-1}\left(r^i + B^{i\mathrm{T}}\left(\Theta^{i+1}\hat{s}^i + \beta^{i+1}\right)\right)$$
$$\Theta^i \leftarrow \left(Q^i + A^{i\mathrm{T}}\Theta^{i+1}A^i\right) + \left(R^i + A^{i\mathrm{T}}\Theta^{i+1}B^i\right)\Gamma^i \quad (i \neq 1)$$
$$\beta^i \leftarrow A^{i\mathrm{T}}\Theta^{i+1}\left(B^i\gamma^i + \hat{s}^i\right) + A^{i\mathrm{T}}\beta^{i+1} + t^i + R^i\gamma^i \quad (i \neq 1)$$

The steps v^i and y^i can then be recovered in the following loop

$$y^1 \leftarrow 0;$$

for $i = 1, \ldots, N$

$$v^i \leftarrow \Gamma^i y^i + \gamma^i$$
$$y^{i+1} \leftarrow A^i y^i + B^i v^i + \hat{s}^i.$$

2.2 Application of the parallel band solver

When the initial value $y^1 = 0$ is included with the system (2.7–2.10), the unknowns are ordered as

$$(q^1, y^1, v^1, q^2, y^2, v^2, \ldots, q^{N+1}, y^{N+1}),$$

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

$$\begin{bmatrix}
0 & -I & & & & & & & & & \\
-I & Q^1 & R^1 & A^{1\mathrm{T}} & & & & & & & \\
 & R^{1\mathrm{T}} & S^1 & B^{1\mathrm{T}} & & & & & & & \\
 & A^1 & B^1 & 0 & -I & & & & & & \\
 & & & -I & Q^2 & R^2 & A^{2\mathrm{T}} & & & & \\
 & & & & R^{2\mathrm{T}} & S^2 & B^{2\mathrm{T}} & & & & \\
 & & & & A^2 & B^2 & 0 & -I & & & \\
 & & & & & & -I & Q^3 & R^3 & A^{3\mathrm{T}} & \\
 & & & & & & & R^{3\mathrm{T}} & S^3 & B^{3\mathrm{T}} & \\
 & & & & & & & A^3 & B^3 & 0 & -I \\
 & & & & & & & & & -I & Q^4 \\
 & & & & & & & & & & \ddots
\end{bmatrix} \qquad (2.17)$$

with a right-hand side of

$$(0, -t^1, -r^1, -\hat{s}^1, -t^2, -r^2, -\hat{s}^2, \ldots, -\hat{s}^N, -t^{N+1}). \tag{2.18}$$

This matrix is symmetric, with dimension $2n(N+1)+mN$, and a half-bandwidth of $(2n+m-1)$. In the neighbourhood of a solution which satisfies Assumption 1, it will be nonsingular.

We now give a brief description in general terms of the algorithm, described in [14], for solving banded linear systems on a P-processor shared-memory multiprocessor. Suppose the system to be solved is

$$Fx = c, \quad F \in R^{M \times M},$$

where F has half-bandwidth k, that is $F_{ij} = 0$ when $|i-j| > k$. We assume impicitly that M is large and k is by comparison very small. The method starts by dissecting F along its diagonal into P large blocks, with each pair of large blocks separated by a smaller $k \times k$ block. Notation for the blocks produced by the dissection is introduced by writing F as:

$$F = \begin{bmatrix}
F^1 & C^{12} & & & & & & & \\
T^{12} & D^1 & T^{21} & & & & & & \\
 & C^{21} & F^2 & C^{22} & & & & & \\
 & & T^{22} & D^2 & T^{31} & & & & \\
 & & & C^{31} & F^3 & C^{32} & & & \\
 & & & & \ddots & \ddots & \ddots & & \\
 & & & & & \ddots & \ddots & \ddots & \\
 & & & & & & T^{P-1,2} & D^{P-1} & T^{P,1} \\
 & & & & & & & C^{P1} & F^P
\end{bmatrix}.$$

Here the main blocks F^j are square, with half-bandwidth k, and have approximately equal dimensions M_P, where $M_P = M/P - k$. The separator blocks D^j are $k \times k$ and dense, while the left and right flanking matrices $C^{j,1}$ and $C^{j,2}$ are $M_P \times k$, and the top and bottom flanking matrices $T^{j,1}$ and $T^{j,2}$ are $k \times M_P$. A symmetric row and column dissection reordering is applied to F, giving

$$P_A F \Pi_A = \left[\begin{array}{ccccc|ccccc}
F^1 & & & & & C^{12} & & & & \\
 & F^2 & & & & C^{21} & C^{22} & & & \\
 & & \ddots & & & & \ddots & \ddots & & \\
 & & & F^{P-1} & & & & C^{P-1,1} & C^{P-1,2} & \\
 & & & & F^P & & & & C^{P1} & \\
\hline
T^{12} & T^{21} & & & & D^1 & & & & \\
 & T^{22} & T^{31} & & & & D^2 & & & \\
 & & \ddots & \ddots & & & & \ddots & & \\
 & & & T^{P-1,2} & T^{P1} & & & & D^{P-1} &
\end{array}\right].$$

Each block F^j is then assigned to a single processor, where it is factorized using an 'enhanced row pivoting' strategy, to give

$$P^j F^j \Pi^j = \begin{bmatrix} L^j & 0 \\ L^j_{(2)} & I \end{bmatrix} \begin{bmatrix} U^j & U^j_{(2)} \\ 0 & V^j \end{bmatrix},$$

where L^j and U^j are square, and lower and upper triangular respectively, while the blocks I and V^j are square with dimension at most $2k$. If F^j is well-conditioned, it is usually the case that $\Pi^j = I$ and that the usual row-partial-pivoting factorization $P^j F^j = L^j U^j$ is obtained.

However, because of the partitioning above, the F^j may be ill-conditioned or even singular (with rank at least $M_P - 2k$), so column pivoting may also be needed for stability.

The U^j factors are also used to eliminate the blocks $T^{j,1}$ and $T^{j,2}$ which now appear in the bottom left of the reordered system, giving rise to multiplier matrices $M^{j,1}$ and $M^{j,2}$ in the overall L factor of F. In addition, more symmetric row and column pivoting is performed on the overall matrix to move the $U^j_{(2)}$ and V^j blocks out of the top left partition. At this point, we have produced a partial LU factorization

$$P_B F \Pi_B = \bar{L}\bar{U},$$

where

$$\bar{L} = \left[\begin{array}{cccccc} L^1 & & & & & \\ & L^2 & & & & \\ & & \ddots & & & 0 \\ & & & \ddots & & \\ & & & & L^P & \\ \hline L^1_{(2)} & & & & & I \\ M_b^{12} & M_b^{21} & & & & \\ & \ddots & & & & \ddots \\ & & & M_b^{P-1,2} & M_b^{P1} & \\ & & & & L^P_{(2)} & I \end{array}\right],$$

$$\bar{U} = \left[\begin{array}{cccccc} U^1 & & & & U^1_{(2)} & \tilde{C}_b^{12} \\ & U^2 & & & & \tilde{C}_b^{21} \\ & & \ddots & & & \ddots \\ & & & \ddots & & \tilde{C}_b^{P-1,2} \\ & & & U^P & \tilde{C}_b^{P1} & U^P_{(2)} \\ \hline 0 & & & & \hat{F} & \end{array}\right].$$

The Schur complement $\hat{F}$ contains fill-in which is generated during the elimination of the $T^{j,1}$ and $T^{j,2}$ blocks, and during the row and column reordering which takes place after factorization of the individual blocks. The remainder of the algorithm consists of factorizing $\hat{F}$ on a single processor, and then recovering the solution using back- and forward-substitution. For further information the reader is referred to [14].

A slightly modified version of this algorithm is used to solve (2.17–2.18). Each separator block D^j is chosen to be

$$D^j = \begin{bmatrix} Q^i & R^i & A^{i\mathrm{T}} \\ R^{i\mathrm{T}} & S^i & B^{i\mathrm{T}} \\ A^i & B^i & 0 \end{bmatrix},$$

for some value of i. The dimension of D^j is $(2n+m)$, rather than the $(2n+m-1)$ which would follow from the direct application of the general bandsolver. However, the blocks $C^{j,1}$, $C^{j,2}$, $T^{j,1}$ and $T^{j,2}$ induced by this choice of D^j contain a number of zero rows and columns, which give rise to computational savings during the factorization process. The total amount of fill-in will be smaller, and the reduced system will have a narrower bandwidth than in the general case.

Parallelism can also be achieved in the solution of the adjoint equation for p (in Step 4 of Algorithm II and Step 5 of Algorithm III). This step can be thought of as the solution of a block-bidiagonal system whose coefficient matrix has the form

$$\begin{bmatrix} I & -A^1 & & & \\ & I & -A^2 & & \\ & & \ddots & & \\ & & & \ddots & -A^N \\ & & & & I \end{bmatrix} \tag{2.19}$$

A similar partitioning into P main blocks is done, with each separator block being one of the $n \times n$ identity matrices on the main diagonal. The algorithm is essentially a simplified version of the parallel band solver. The same idea (for solving systems of the form (2.19) on message-passing architectures, where the submatrices are scalars) appears in [11].

2.3 Numerical results

Parallelized versions of the above algorithm were implemented on the Alliant FX/8 multiprocessor at Argonne National Laboratory. Here we give some specifics of the implementations, and some numerical results on test problems from the literature.

Possibilities for parallelism in the linear algebra steps of the three algorithms have already been discussed. The other main issue is parallelization of the function evaluations. For Algorithms I and II this is trivial. A single routine to find the values of f^i, A^i, B^i, t^i, r^i, Q^i, R^i and S^i for a given value of i is all that is needed for Algorithm I. In our implementations, the index set $\{1, 2, \ldots, N\}$ is broken up into P equally-sized partitions (P is the number of processors), and each processor does the evaluations corresponding to one of the partitions. For Algorithm II, t^i and A^i must be evaluated separately, in Step 3. Here two evaluation routines are needed, but again parallel implementation for different values of i is possible. It may be desirable to store partial results from Step 3 for use in Step 1 of the following iteration.

Algorithm III (Newton's method) has the same evaluation pattern as Algorithm II, with the important difference that in Step 3 it is generally necessary to evaluate the functions f^i *in sequence*. The effect on execution time depends on the cost of evaluating f^i relative to other evaluations, and the overall cost of the evaluations relative to the linear algebra.

We report on timings obtained for the following implementations of Algorithms I, II and III on the Alliant FX/8. (For the multiprocessor algorithms, the functions and derivatives are evaluated concurrently, with the exceptions noted.)

Algorithm I: (2.7–2.10) solved with the parallel bandsolver, on 8 processors.

Algorithm II: (2.7–2.10) solved using the Dunn and Bertsekas recurrences, on 8 processors.

Algorithm III: Dunn and Bertsekas recurrences, with *sequential* evaluation of the f^i, on 8 processors.

Algorithm III(S): same as III, on a single processor (i.e. the standard Dunn and Bertsekas implementation of Newton's method).

Algorithm III-P(S): same as III(S), except that the Polak's method is used.

These algorithms were tested on number of problems arising from the literature:

Problem I (Bertsekas [1]): ($m = 1$, $n = 2$) Choosing $h = 1/N$,

$$\min \sum_{i=1}^{N} 5(u^i)^2 + 2\left(x_1^{i+1}\right)^2 + \left(x_2^{i+1}\right)^2$$

$$\text{s.t. } x^{i+1} = \begin{bmatrix} 1 & h \\ -h & 1 \end{bmatrix} x^i + \begin{bmatrix} 0 \\ h \end{bmatrix} u^i, \quad i = 1, \ldots, N.$$

(Here subscripts indicate components of x^i.)

Problem 2 (Kelley and Sachs [8]): ($m = 1$, $n = 1$) This problem is a discretization of the continuous problem

$$\min_{u(t)} \int_0^{0.3} 0.5e^{-t}\left((x(t) - 1.5)^2 + (u(t) - 3)^2\right) + \frac{u^4(t)}{40}\, dt$$

$$\text{s.t. } \dot{x}(t) = u(t) - t^2, \quad x(0) = 1.$$

Given a value of N, set $h = 0.3/N$ and look for u^i to approximate $u((i - 0.5)h)$ and x^i to approximate $x((i - 1)h)$. The integral can be replaced by a trapezoidal rule, and the state equation by the midpoint rule approximation

$$x^{i+1} = x^i + hu^i - h^3(i - 0.5)^2.$$

Problem 3 (Di Pillo, Grippo and Lampariello [2]): ($m = 1$, $n = 2$) In the terminology of (1.1, 1.2), and choosing $h = 1/N$,

$$l_1(x^1, u^1) = \frac{5h}{2}\left[(x_1^1)^2 + (x_2^1)^2\right] + 5h(u^1)^2$$

$$l_{N+1}(x^N) = \frac{5h}{2}\left[(x_1^{N+1})^2 + (x_2^{N+1})^2\right]$$

$$l_i(x^i, u^i) = 5h\left[(x_1^i)^2 + (x_2^i)^2 + (u^i)^2\right]$$

$$f^i(x^i, u^i) = \begin{bmatrix} x_1^i + 5h\left[\left(1 - (x_2^i)^2\right)x_1^i - x_2^i + u^i\right] \\ x_2^i + 5hx_1^i \end{bmatrix}.$$

Problem 2A: ($m = 1$, $n = 1$) This is the same as Problem 2, except that each function and derivative evaluation is done 100 times. Hence the computation time is (artificially) dominated by evaluations rather than by the linear algebra.

The timings for a single iteration with for N set to 1000, are shown in *Table 1*. For the first three problems, the cost of the linear algebra tends to dominate. This can be deduced by comparing the times for the Dunn–Bertsekas-based algorithms II (fully concurrent evaluations) and III(S) (fully serial evaluations). In going from Algorithm II to Algorithm I, the speedup of about 2 is entirely due to the use of the parallel bandsolver. This speedup is similar to the relative times for an 8-processor implementation of the parallel bandsolver and one-processor Gaussian elimination, as obtained in [14]. (The computational overhead in operating on the partitioned system is quite high, so that although the parallel algorithm makes efficient use of the multiple processors, it does not attain a linear speedup relative to the best serial algorithm. This is discussed in detail in [14].) Slightly better speedup is obtained for Problem 2, in which the system (2.17–2.18) has smaller bandwidth than in Problems 1 and 3, and again this is consistent with the results in [14].

Table 1
Times for one iteration of each algorithm on the Alliant FX/8 in seconds, $N = 1000$.

	Prob. 1	Prob. 2	Prob. 3	Prob. 2A
Alg. I	0.382	0.155	0.387	1.24
Alg. II	0.730	0.352	0.731	1.44
Alg. III	0.731	0.407	0.733 *	5.54
Alg. III(S)	0.765	0.462	0.776 *	9.02
Alg. III-P(S)	1.46	0.655	1.47 *	9.21

* = failed to converge.

In Problem 2A, the function evaluations dominate the computation, and we see a progressive improvement in going from the serial algorithm III(S), to the 'partially parallel' III, to the 'fully parallel' II. The advantage gained by using the parallel band solver (Algorithm I) is smaller in a relative sense than in the other problems. However, it is clear by comparing Algorithm I with Algorithm III(S) that a highly efficient 8-processor implementation has been achieved.

Im summary, on an 8-processor machine, we can obtain an algorithm which is two to three times faster than the best serial implementation of Newton's method, for problems in which the linear algebra dominates, and almost eight times faster than Newton's method, when the function evaluations dominate.

We stress again that only the 'no-frills' locally convergent algorithms have been implemented above. For practical codes, it will be necessary to include features which ensure global convergence. Also, it may be desirable to make use of the chord method, in which the second-derivative matrices are not calculated (and factorized) on every iteration.

3. The constrained problem

3.1 Active set methods

Inclusion of the general constraints (1.3) gives a more difficult problem, which can again be solved in a sequential quadratic programming framework. We sketch the approach in this section. There are two variants of SQP for handling inequality constraints, known in the literature as 'IQP' and 'EQP'. In IQP, the constraints (1.3) are linearized at each step, and the quadratic subproblem (2.15) is solved subject to the additional constraints

$$g^i(x^i, u^i) + y^i \frac{\partial}{\partial x} g^i(x^i, u^i) + v^i \frac{\partial}{\partial u} g^i(x^i, u^i) \leqslant 0, \quad i = 1, \ldots, N. \tag{3.1}$$

The other terms r^i, t^i, Q^i, R^i, and S^i in (2.15) need to be redefined in terms of a Lagrangian with an additional term:

$$\mathscr{L}(u, x, p, \mu) = l_{N+1}(x^{N+1}) + \sum_{i=1}^{N} l_i(x^i, u^i) - \sum_{i=1}^{N} \left(x^{i+1} - f^i(x^i, u^i)\right) p^{i+1} + \sum_{i=1}^{N} g^i(x^i, u^i)\mu^i.$$

The quadratic programming problem (2.15, 3.1) can be solved using an active set approach, where at each minor iteration a subset of the components of (3.1) are chosen to hold at equality, and the remainder are ignored, at least until a line search is done. Hence (2.15) may be repeatedly solved subject to constraints of the form

$$\begin{aligned} &g^i_j(x^i, u^i) + y^i \frac{\partial}{\partial x} g^i_j(x^i, u^i) + v^i \frac{\partial}{\partial u} g^i_j(x^i, u^i) = 0, \\ &j \in \Lambda_i \subset \{1, \ldots, \nu_i\}, \quad i = 1, \ldots, N. \end{aligned} \tag{3.2}$$

Writing down the first order conditions for this subproblem, we obtain a set of linear equations in v^i, y^i, q^i, and μ^i_j ($j \in \Lambda_i$) which is obtained by augmenting (2.7–2.10) with appropriate terms in μ^i, and including the equations (3.2). Again these equations can be solved via recurrence relations (see Ohno [9], or Dunn [3] and Pantoja and Mayne [10] for the case in which the g^i are functions of u^i only). Alternatively, after an appropriate ordering of equations and variables, a block-banded system is again obtained, which can again be solved with the parallel band solver.

In the EQP approach, the active constraint sets Λ_i are fixed during each outer iteration, rather than at each minor iteration as in IQP. The innermost iteration is identical in form to that of IQP, namely (2.15, 3.2), and hence can be solved as before. The relative merits of IQP and EQP have been extensively discussed elsewhere (see for example Powell [13]).

We do not pursue these approaches further here, since they are often unsuitable for large-scale problems. The reason for this is that the active set $\Lambda = \{\Lambda_1, \ldots, \Lambda_N\}$ is generally only allowed to change by a small number of components at each (major or minor) iteration. Hence if the initial estimate of Λ is far from the optimal active set, many iterations will be required.

Once the correct active constraint manifold has been identified, these algorithms are R-quadratically convergent in u.

3.2 The simple control-constrained case – a gradient projection method

An alternative to active set methods, which is attractive when the feasible region defined by the constraints (1.3) is convex and has simple geometry, is the two-metric gradient projection algorithm (see Bertsekas [1], Gafni and Bertsekas [6] and Dunn [3]). This method has the advantage that large changes to the active set are allowed at each iteration, while it retains the good local convergence properties of the active set algorithms.

We sketch this approach for the important special case in which the constraints have the form (1.4) (see Bertsekas [1] for the details). The problem (1.1, 1.2) is regarded as a function of u alone (i.e. $F(u)$). As in Algorithm III of Section 2, x is obtained from the state equations (1.2) and p is obtained from the adjoint equation (2.3), so that in (2.15), $t^i = 0$ and $\hat{s}^i = 0$. At each iteration, an 'almost-active set' $\bar{\Lambda} = \{\bar{\Lambda}_1, \ldots, \bar{\Lambda}_N\}$ is constructed, where $\bar{\Lambda}_i$ contains the indices of the components of u^i which at or near their bound $\pm\eta$, and for which the corresponding component of $\nabla F(u)$ has the "correct" sign (i.e. $\partial F/\partial u^i_j > 0$ if $u^i_j \approx -\eta$ and $\partial F/\partial u^i_j < 0$ if $u^i_j \approx \eta$). (2.15) is then solved, using the definitions of Q^i, R^i, S^i and r^i from (2.1) and (2.6), subject to the additional constraints

$$v^i_j = 0, \quad j \in \bar{\Lambda}_i, \quad i = 1, \ldots, N. \tag{3.3}$$

This process yields the 'Newton' part of the step. The 'first-order' part can be obtained by setting

$$v^i_j = -\sigma^i_j \frac{\partial F}{\partial u^i_j}, \quad j \in \bar{\Lambda}_i, \quad i = 1, \ldots, N$$

(where $\underline{\sigma} \leqslant \sigma^i_j \leqslant \bar{\sigma}$, $\underline{\sigma}$ and $\bar{\sigma}$ being positive constants; we choose $\sigma^i_j \equiv 1$). A search is then carried out along the piecewise-linear path $P(u + \alpha v)$, $\alpha \in [0, 1]$, where P denotes projection of a vector in R^{km} onto the box $\{w \in R^{km} \mid w_i \in [-\eta, \eta]\}$. An Armijo linesearch strategy is used, that is, the values $\alpha = 1, \beta, \beta^2, \ldots (\beta < 1)$ are tried, until a sufficient decrease criterion is met. For this value of α, u is reset to $P(u + \alpha v)$ for the next iteration. The values of x and p are recalculated accordingly.

Solution of the constrained problem (2.15, 3.3) can be done using a modified version of the recurrences in Algorithm DB (see Dunn [3]), which we state below. Let P^i be the matrix

$$P^i = \left[e^{\mathrm{T}}_j\right]_{j \notin \bar{\Lambda}_i},$$

where e_j is the jth column of the $m \times m$ identity matrix. Also for any $m \times m$ matrix S, denote $P^i S P^{i\mathrm{T}}$ by $[S]_{ii}$.

Algorithm DB(C)

$$\Theta^{N+1} = Q^{N+1}; \ \beta^{N+1} = t^{N+1};$$

for $i = N, \ldots, 1$

$$\Gamma^i \leftarrow -\left[S^i + B^{i\mathrm{T}}\Theta^{i+1}B^i\right]_{ii}^{-1} P^i\left(R^{i\mathrm{T}} + B^{i\mathrm{T}}\Theta^{i+1}A^i\right)$$

$$\gamma^i \leftarrow -\left[S^i + B^{i\mathrm{T}}\Theta^{i+1}B^i\right]_{ii}^{-1} P^i\left(r^i + Bi^{\mathrm{T}}\left(\Theta^{i+1}\hat{s}^i + \beta^{i+1}\right)\right)$$

$$\Theta^i \leftarrow \left(Q^i + A^{i\mathrm{T}}\Theta^{i+1}A^i\right) + \left(R^i + A^{i\mathrm{T}}\Theta^{i+1}B^i\right)P^{i\mathrm{T}}\Gamma^i \quad (i \neq 1)$$

$$\beta^i \leftarrow A^{i\mathrm{T}}\Theta^{i+1}\left(B^i P^{i\mathrm{T}}\gamma^i + \hat{s}^i\right) + A^{i\mathrm{T}}\beta^{i+1} + t^i + R^i P^{i\mathrm{T}}\gamma^i \quad (i \neq 1)$$

The steps $P^i v^i$ and y^i can be recovered as in the loop after Algorithm DB.

An alternative method for solving (2.15, 3.3) is to use a reduced version of the linear system (2.17, 2.18). We can simply delete rows and columns of this matrix which correspond to the indices $j \in \bar{A}_i$, $i = 1, \ldots, N$. The resulting coefficient matrix will have dimension between $2n(N+1)$ and $2n(N+1) + mN$. In our implementation we opt for the simpler approach of 'eliminating' the variables by replacing the rows and columns in question by zeros (except for a one on the diagonal) and by inserting a zero in the appropriate right-hand side position. This is done because (a) we wish to avoid disturbing the special structure of (2.17) and its associated computational savings, (b) the overhead involved in compressing the elements of (2.17) to give a reduced system would be substantial, and (c) when m and n are similar, the potential reduction in the size of the matrix is not very great.

3.3 Numerically results

We report on results for three implementations of the algorithm described above. In all cases, since gradient projection is being applied to F as a function of u alone, the states x^i must be eliminated using the state equations (1.2), that is, the functions f^i must be evaluated in sequence.

Algorithm GP(B): Step $P^i v^i$ obtained by solving (2.17–2.18) (modified by (3.3)) using the parallel bandsolver. All functions except f^i and l_i evaluated concurrently on 8 processors.

Algorithm GP: Step $P^i v^i$ obtained using Algorithm DB(C). All functions except f^i and l_i evaluated concurrently on 8 processors.

Algorithm GP(S): One-processor implementation of Algorithm GP.

Results are obtained for four problems. All algorithms gave an identical sequence of iterates, and were terminated when the projected gradient norm fell below 10^{-10}. In all cases $N = 1000$, and a starting point of $u = 0$ was used.

Problem 1 from Section 2.3, with $\eta = 1$. Converges in 3 iterations, with 183 components of u active.

Problem 2 from Section 2.3, with $\eta = 2$. Converges in 3 iterations, with 740 components of u active.

Problem 2 from Section 2.3, with $\eta = 3$. Converges in 5 iterations, with no components of u active.

Problem 2B with $\eta = 2$. Same as problem 2 from Section 2.3, except that function evaluation costs are artificially magnified by doing each evaluation (except for f^i and l_i) 100 times.

Total times to solution on the Alliant FX/8 are given in *Table 2*. Again, the cost of the linear algebra dominates in the first three columns. The use of the parallel band solver gives a

Table 2
Times to solution on the Alliant FX/8 in seconds, $N = 1000$.

	Prob. 1 ($\eta = 1$)	Prob. 2 ($\eta = 2$)	Prob. 3 ($\eta = 3$)	Prob. 2B ($\eta = 2$)
Alg. GP(B)	1.69	1.01	1.55	2.68
Alg. GP	2.42	1.29	2.67	2.91
Alg. GP(S)	2.59	1.52	3.01	14.4

speedup of somewhat less than that obtained for the corresponding unconstrained problems. This can be attributed to the fact (mentioned above) that the band solver in Algorithm GP(P) does not at present take advantage of the reduced dimension of the Newton step. (Hence the fewer constraints active, the better the speedup – compare the second and third columns.) Best results are obtained for Problem 2B, which is meant to typify applications in which the state equations f^i are easily evaluated (e.g. linear), but evaluation of the objective functions and their derivatives is expensive. A speedup of 5.4 is obtained by the best parallel algorithm GP(B) over the best sequential algorithm GP(S).

3.4 A hybrid approach

As noted above, the need for evaluation of the nonlinear state equations (1.2) between iterations of the gradient projection method presents a possible serial bottleneck in the simple control-constrained case, since this entails evaluating the functions f^i in sequence. A potentially more parallelizable approach would be to use a two-level method. Sequential quadratic programming (IQP variant) could be applied to the original nonlinear problem (1.1, 1.2, 1.4), yielding a subproblem in which the state equations are linear, and in which the bound constraints (1.4) still appear. This subproblem could then be solved using a gradient projection method. The state equations still need to be evaluated between each gradient projection iteration, but since they are now linear, this step can be implemented in parallel using much the same techniques which were used to parallelize the banded system solver.

4. Conclusions and further work

We have discussed multiprocessor implementations of some basic rapidly-convergent algorithms for discrete-time optimal control problems, which attempt to exploit parallelism both in the linear algebra and in evaluation of the functions. Further work remains to be done to implement these methods on message-passing machines (e.g. hypercubes and massively parallel architectures). Additionally, algorithms with better global convergence properties are needed. Simple techniques for ensuring global convergence of Newton's method and gradient projection methods are quite well understood; for SQP more complicated devices are needed. The challenge is to implement the globalization strategies without interfering too much with the parallelism which has been obtained in the basic algorithms above.

References

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

[2] G. Di Pillo, L. Grippo and F. Lampariello, A class of structured quasi-Newton algorithms for optimal control problems, in: *Proc. IFAC Applications of Nonlinear Programming to Optimization and Control* (1983).

[3] J. Dunn, A projected Newton method for minimization problems with nonlinear inequality constraints, *Numer. Math.* 53 (1988) 377–409.

[4] J. Dunn and D.P. Bertsekas, Efficient dynamic programming implementations of Newton's method for unconstrained optimal control problems, *J. Optim. Theory Appl.*, 63 (1989) 23–38.

[5] R. Fletcher, *Practical Methods of Optimization (2nd edition)* (Wiley, New York, 1987).

[6] E.M. Gafni and D.P. Bertsekas, Two-metric projection methods for constrained optimization, *SIAM J. Control Optim.* 22 (1984) 936–964.

[7] D.H. Jacobson and D.Q. Mayne, *Differential Dynamic Programming* (American Elsevier, New York, 1970).

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

[9] K. Ohno, A new approach to differential dynamic programming for discrete time systems, *IEEE Trans. Automat. Control* AC-23 (1978) 37–47.
[10] J.F.A. Pantoja and D.Q. Mayne, A sequential quadratic programming algorithm for discrete optimal control problems with control inequality constraints, submitted, 1988.
[11] L.J. Podrazik and G.G.L. Meyer, A parallel first-order linear recurrence solver, *J. Parallel Dist. Comp.* 4 (1987) 117–132.
[12] E. Polak, *Computational Methods in Optimization* (Academic Press, New York, 1970).
[13] M.J.D. Powell, ed., *Nonlinear Optimization 1981* (Academic Press, New York, 1982).
[14] S.J. Wright, Parallel algorithms for banded linear systems, Argonne National Laboratory Preprint MCS-P64-0289, 1989, SIAM J. Sci. Statist. Comput., to appear.