PARALLEL AND VECTOR COMPUTATION FOR STOCHASTIC OPTIMAL CONTROL APPLICATIONS

F. B. HANSON
Department of Mathematics, Statistics, and Computer Science
University of Illinois at Chicago
P. O. Box 4348; M/C 249
Chicago, IL 60680
3rd CACC

Abstract. A general method for parallel and vector numerical solutions of stochastic dynamic programming problems is described for optimal control of general nonlinear, continuous time, multibody dynamical systems, perturbed by Poisson as well as Gaussian random white noise. Possible applications include lumped flight dynamics models for uncertain environments, such as large scale and background random atmospheric fluctuations. The numerical formulation is highly suitable for a vector multiprocessor or vectorizing supercomputer, and results exhibit high processor efficiency and numerical stability. Advanced computing techniques, data structures, and hardware help alleviate Bellman's *curse of dimensionality* in dynamic programming computations.

1. Introduction. The primary motivation for this research is to provide a provide a general computational treatment of stochastic optimal control applications in continuous time. In addition, fast and efficient methods are being developed by the optimization of stochastic dynamic programming algorithms for larger multibody problems. The optimization will help alleviate *Bellman's curse of dimensionality*, in that the computational problem greatly increases as the dimension of the state space increases. Optimization consists of parallelization and vectorization techniques to enhance performance on advanced computers, such as parallel processors and vectorizing supercomputers. General Markov random noise in continuous time consists of two kinds, Gaussian and Poisson. Gaussian white noise, being continuous but nonsmooth, is used to model background random fluctuations, such as turbulence and external field variations. Poisson white noise (its frequency spectrum is also flat like Gaussian noise), being discontinuous, is useful for modeling large random fluctuations, such as shocks, collisions, unexpected external events and large environmental changes. Our general feedback control approach combines the treatment of both linear and nonlinear (i.e., singular and nonsingular) control through the use of small and non-small quadratic costs. The methods also handle deterministic and stochastic control in the same code, making it convenient for checking the effects of stochasticity on the application. Some actual applications are models of resources in an uncertain environment [15], [11], [8]. Some potential applications are flight dynamics under random wind conditions [2] and other resource models [12].

The Markov, multibody dynamical system is illustrated in Figure 1 and is governed by the stochastic differential equation (SDE):

$$d\mathbf{y}(s) \;=\; \mathbf{F}(\mathbf{y},s,\mathbf{u})ds \;+\; G(\mathbf{y},s)d\mathbf{W}(s) \;+\; H(\mathbf{y},s)d\mathbf{P}(s)\;, \qquad (1.1)$$

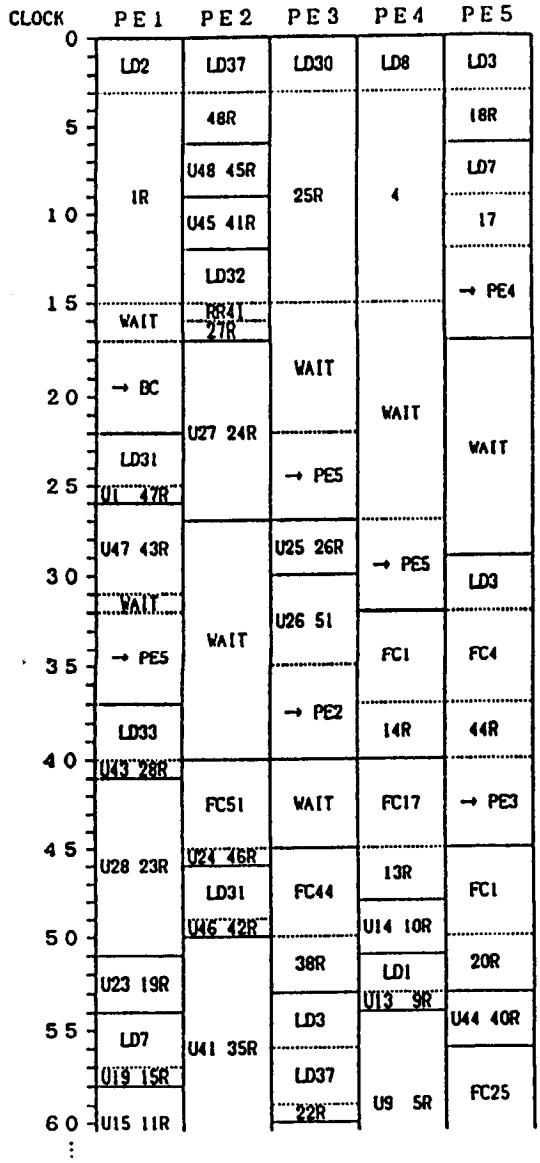

Fig.8 Execution image of machine codes on OSCAR.

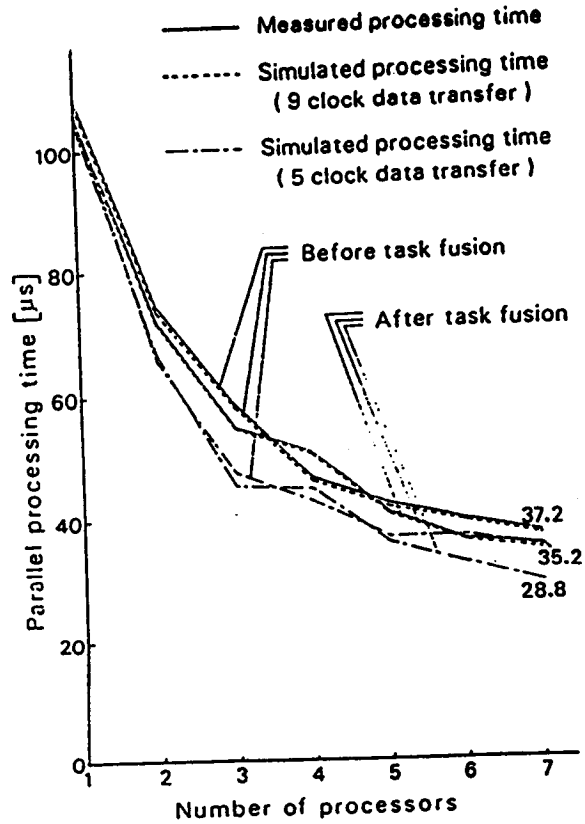

Fig.9 Parallel processing time measured on OSCAR and simulated parallel processing time.

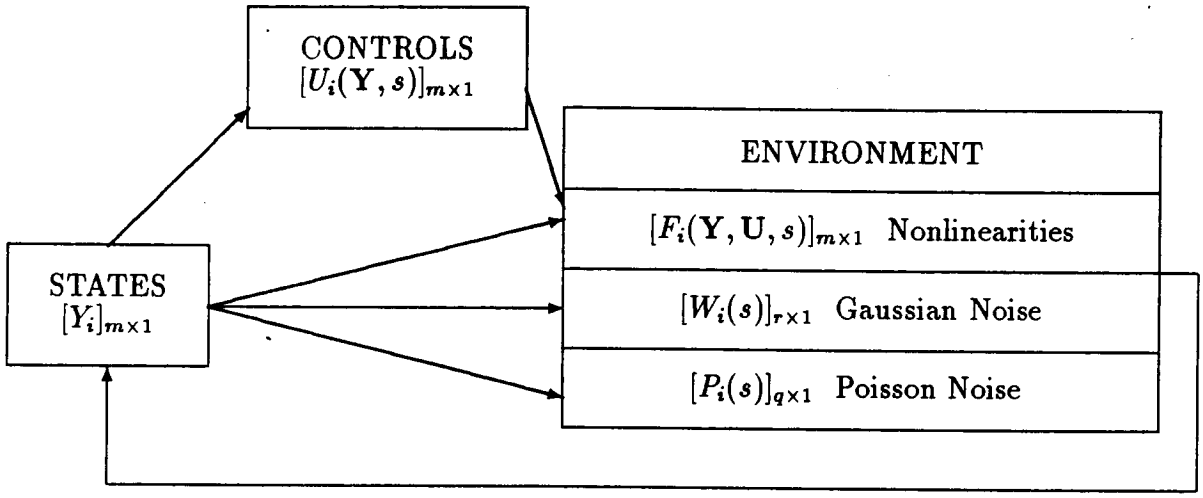

Figure 1: The multibody dynamical system.

$$\mathbf{y}(t) = x \; ; \; 0 < t < s < t_f \; ; \; \mathbf{y}(s) \in \mathcal{D}_y \; ; \; \mathbf{u} \in \mathcal{D}_u \; ,$$

where $\mathbf{y}(s)$ is the $m \times 1$ multibody state vector at time s starting at time t, $\mathbf{u} = \mathbf{u}(\mathbf{y}, s)$ is the $n \times 1$ feedback control vector, $\mathbf{W}$ is the r-dimensional normalized Gaussian white noise vector, $\mathbf{P}$ is the independent q-dimensional Poisson white noise vector with jump rate vector $[\lambda_i]_{q\times 1}$, $\mathbf{F}$ is the $m \times 1$ deterministic nonlinearity vector, G is an $m \times r$ diffusion coefficient array, and H is an $m \times q$ Poisson amplitude coefficient array.

The control criterion is the optimal expected cost performance,

$$V^*(\mathbf{x}, t) = \min_{\mathbf{u}} \left[MEAN_{\mathbf{P},\mathbf{W}} \left[V[\mathbf{y}, s, \mathbf{u}, \mathbf{P}, \mathbf{W}] \mid \mathbf{y}(t) = \mathbf{x} \right] \right] , \tag{1.2}$$

over some specified optimal control set $\mathcal{D}_u$, where the total cost is

$$V[\mathbf{y}, t, \mathbf{u}, \mathbf{P}, \mathbf{W}] = \int_t^{t_f} ds \; C(\mathbf{y}(s), s, \mathbf{u}(\mathbf{y}(s), s)) \; , \tag{1.3}$$

on the time horizon $(t, \; t_f)$. In (1.3), the instantaneous cost function $C = C(\mathbf{x}, t, \mathbf{u})$ is assumed to be a quadratic function of the control,

$$C(\mathbf{x}, t, \mathbf{u}) = C_0(\mathbf{x}, t) + \mathbf{C}_1^T(\mathbf{x}, t)\mathbf{u} + \tfrac{1}{2}\mathbf{u}^T C_2(\mathbf{x}, t)\mathbf{u} \; . \tag{1.4}$$

The unit cost of the control increases with $\mathbf{u}$ when C_2 is positive definite. For example, the cost criterion could be minimal fuel consumption, minimum distance to target or minimum time to target. No final salvage value is assumed at final time, so V is zero at $t = t_f$.

In addition, the deterministic, nonlinear dynamics in (1.1) are assumed to be linear in the controls,

$$\mathbf{F}(\mathbf{x}, t, \mathbf{u}) = \mathbf{F}_0(\mathbf{x}, t) + F_1(\mathbf{x}, t)\mathbf{u} \; , \tag{1.5}$$

but nonlinear in the multibody state variable $\mathbf{x}$.

For numerical purposes, it is more convenient to convert equations (1.1)-(1.2) to an effectively deterministic partial differential equation using Bellman's of optimality as illustrated in the optimization step from optimal control vector $\mathbf{U}^*$ to optimal expected costs V^* in Fig. 2. The Bellman functional PDE of stochastic dynamic programming,

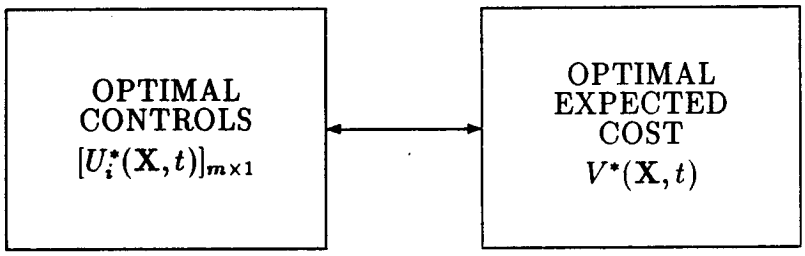

Figure 2: The optimization step from controls to costs.

$$
\begin{aligned}
0 \;=\; & V_t^* \;+\; L[V^*] \;\equiv\; V_t^* \;+\; \mathbf{F}_0^T \nabla V^* \;+\; \tfrac{1}{2} G G^T(\mathbf{x},t) : \nabla\nabla^T V^* \\
& +\; \sum_{l=1}^{q} \lambda_l \cdot [\; V^*(\mathbf{x} + \mathbf{H}_l(\mathbf{x},t),t) - V^*(\mathbf{x},t) \;] \qquad (1.6) \\
& +\; C_0 \;+\; (\tfrac{1}{2}\mathbf{U}^* - \mathbf{U}_R)^T C_2 \mathbf{U}^* \;,
\end{aligned}
$$

follows from the generalized *Itô* chain rule for Markov SDEs as in [7] and [15], where $\mathbf{U}^*$ is the optimal feedback control computed by constraining the unconstrained or regular control,

$$
\mathbf{U}_R(\mathbf{x},t) \;=\; -C_2^{-1}(\mathbf{C}_1 \;+\; {F_1}^T \nabla V^*) \;, \qquad (1.7)
$$

to the control set $\mathcal{D}_u$. In general, the Bellman equation (1.6) is nonlinear with discontinuous coefficients due to the last term, $(\frac{1}{2}\mathbf{U}^* - \mathbf{U}_R)^T C_2 \mathbf{U}^*$, in (1.6) and due to the compact relationship between the constrained, optimal control and the unconstrained, regular control,

$$
U_i^*(\mathbf{x},t) = \min[U_{\max,i}, \min[U_{\min,i}, U_{R,i}(\mathbf{x},t)]], \qquad (1.8)
$$

for $i = 1$ to n controls, where $\mathbf{U}_{\min}$ is the minimum control constraint vector and $\mathbf{U}_{\max}$ is the maximum. As the constraints are attained, the optimal control $\mathbf{U}^*$, changes from the regular control, $\mathbf{U}_R$, to the single bang control values, $\mathbf{U}_{min}$ or $\mathbf{U}_{max}$, which in general are functions of state and time. In (1.6), the symbol (:) denotes the scalar matrix product $A : B = \sum_{i=1}^{m} \sum_{j=1}^{m} A_{ij} B_{ij}$, assuming B is symmetric. It is important to note that the principal equation, the Bellman equation (1.6), is an exact equation for the optimal expected value V_* and does not involve any sampling approximations such as the use of random number generators in simulations.

Since there is no final salvage value and since (1.6) is a backward equation (unlike the usual diffusion equation, which is a forward equation), the final condition is that $V^*(x,t_f) = 0$ using (1.2) and (1.3). On the other hand, boundary conditions for the PDE of stochastic dynamic programming (1.6) are not as simple or as straightforward to state. This is because the boundary conditions vary significantly with the form the deterministic linearity function $\mathbf{F}$, the Gaussian noise W, and the Poisson noise P. Thus for treatment of general boundary conditions, it is most practical to directly integrate (1.6) for the special values of x, or to use the objective functional directly as defined in (1.2) and (1.3). The problem with boundary conditions is also present in stochastic application in continuous time, even when there is no control variable or optimization in the problem.

As the number of multibody state variables, m, increases, the spatial dimension rises, and computational difficulties are present that can compare to those of three-dimensional fluid dynamics computations. This is the famous Bellman's *curse of dimensionality* [3]. Thus there is a great need to make use of advanced-architecture computers, to use parallelization as well as vectorization. The Panel on Future Directions in Control Theory [6] stresses the importance of making gains in such areas as nonlinear control, stochastic control, optimal feedback control and computational methods for control. This paper is a preliminary report on our efforts to treat all of the above mentioned areas combined from the computational point of view.

2. Numerical Methods. The integration of the Bellman equation (1.6) is backward in time, because V^* is specified finally at the final time $t = t_f$, rather than at the initial time. A summary of the discretization in state and backward time is given below:

$$\begin{aligned}
\mathbf{x} &\longrightarrow \mathbf{X_j} = [X_{ij_i}]_{m\times 1} = [X_{i1} + (j_i - 1)\cdot DX_i]_{m\times 1}\,, \\
\mathbf{j} &= [j_i]_{m\times 1}\,, \quad \text{where } j_i = 1 \text{ to } M_i\,, \quad \text{for } i = 1 \text{ to } m\,; \\
t &\longrightarrow T_k = t_f - (k-1)\cdot DT\,, \quad \text{for } k = 1 \text{ to } K\,; \\
V^*(\mathbf{X_j}, T_k) &\longrightarrow V_{\mathbf{j},k}\,; \quad L[V^*](\mathbf{X_j}, T_{k+\frac{1}{2}}) \longrightarrow L_{\mathbf{j},k+\frac{1}{2}}\,;
\end{aligned} \tag{2.1}$$

where DX_i is the mesh size for state i and DT is the step size in backward time.

The numerical algorithm is a modification of the predictor corrector, Crank Nicolson methods for nonlinear parabolic PDEs in [5]. Modifications are made for the switch term and delay term calculations. Derivatives are approximated with an accuracy that is second order in the local truncation error, at all interior and boundary points. The Poisson induced functional or delay term, $V^*(\mathbf{x} + \mathbf{H}_l, t)$, changes the local attribute of the usual PDE to a global attribute, such that the value at a node $[\mathbf{X} + \mathbf{H}_l]_\mathbf{j}$ will in general not be a node. Linear interpolation, with special handing of point near the boundaries, maintains the numerical integrity compatible with the numerical accuracy of the derivative approximations. Even though the Bellman equation (1.6) is a single PDE, the process of solving it not only produces the optimal expected value V^*, but also the optimal expected control law U^*. This is because the PDE is a functional PDE, in which the computation of the regular control is fed back into the optimal value and the optimal value feeds back into regular control through its gradient. The nonstandard part of the algorithm is to decompose this tightly coupled analytical feedback so that both the value and the control can be calculated by successive

iterations, such that each successive approximation of one improves the next approximation of the other. While our procedure may look superficially like a standard application of finite differences, it is not due to the nonstandard features mentioned above. For these reasons, we are not aware of any other successful stochastic dynamic programming code that treats anywhere near the generality of applications that we treat. Variations of this algorithm have been successfully utilized in [15] and [8].

Prior to calculating the values, $V_{\mathbf{j},k+1}$, at the new $(k+1)^{st}$ time step for $k = 1$ to $K-1$, the old values, $V_{\mathbf{j},k}$ and $V_{\mathbf{j},k-1}$, are assumed to be known, with $V_{\mathbf{j}0} \equiv V_{\mathbf{j}1}$. The algorithm begins with an *extrapolator (x) start*:

$$V^{(\mathbf{x})}_{\mathbf{j},k+\frac{1}{2}} = \tfrac{1}{2}(3 \cdot V_{\mathbf{j},k} - V_{\mathbf{j},k-1}) , \tag{2.2}$$

which are then used to compute updated values of the gradient of V^*, the second order derivatives, Poisson functional terms (V^* at $(\mathbf{x}+\mathbf{H})$), regular controls $\mathbf{U}_R$, optimal controls $\mathbf{U}^*$, and finally the new value of the Bellman equation spatial functional $L_{\mathbf{j},k+0.5}$. The extrapolation step greatly speeds up the convergence of the corrector step, except at the initial step. These evaluations are used in the *extrapolated predictor (xp) step*:

$$V^{(\mathbf{xp})}_{\mathbf{j},k+1} = V_{\mathbf{j},k} + DT \cdot \tfrac{1}{2} L^{(\mathbf{x})}_{\mathbf{j},k+\frac{1}{2}} . \tag{2.3}$$

which are then used in the *predictor evaluation (xpe) step*:

$$V^{(\mathbf{xpe})}_{\mathbf{j},k+\frac{1}{2}} = \tfrac{1}{2}(V^{(\mathbf{xp})}_{\mathbf{j},k+1} + V_{\mathbf{j},k}) , \tag{2.4}$$

an approximation which preserves numerical accuracy and which is used to evaluate all terms comprising $L_{\mathbf{j},k+0.5}$. The evaluated predictions are used in the *corrector (xpec) step*:

$$V^{(\mathbf{xpec},\gamma+1)}_{\mathbf{j},k+1} = V_{\mathbf{j},k} + DT \cdot L^{(\mathbf{xpe},\gamma)}_{\mathbf{j},k+\frac{1}{2}} \tag{2.5}$$

for $\gamma = 0$ to γ_{max} while stopping criterion unmet, with *corrector evaluation (xpece) step*:

$$V^{(\mathbf{xpece},\gamma+1)}_{\mathbf{j},k+\frac{1}{2}} = \tfrac{1}{2}(V^{(\mathbf{xpec},\gamma+1)}_{\mathbf{j},k+1} + V_{\mathbf{j},k}) . \tag{2.6}$$

The predicted value is taken as the zeroth correction. The stopping criterion for the corrections is a heuristically motivated comparison to a predictor corrector convergence criterion for a linearized, constant coefficient PDE [13]. The stopping criterion is computed with a robust mesh selection method, so that only a few corrections are necessary. The selection of the mesh ratio, the ratio of the time step DT to the norm of the space or state step $D\mathbf{X}$, guarantees that the corrections will converge whether the Bellman equation (1.6) is parabolic-like (with Gaussian noise) or hyperbolic-like (without Gaussian), according to whether or not an explicit second derivative is in the equation.

Parallelization and vectorization of the algorithm was done by what might be called the "Machine Computational Model Method," i.e., tuning the code to optimizable constructs

that are automatically recognized by the compiler, with the Alliant FX/8 vector multiprocessor [1] in mind. All inner double loops were reordered to fit the Alliant *concurrent outer - vector inner* (*COVI*) model. All non-short single loops were made *vector-concurrent* Short loops became *scalar-concurrent* only. Multiple nested loops were reordered with the two largest loops innermost. A total of 37 out of 39 loops was optimized. Detailed results for a two-state and two-control model with Poisson noise are reported in [9]. Very similar techniques work for the vectorizing Cray supercomputers, except that only inner loops are vectorized. Vectorizing and parallelizing techniques are very similar, because vectorization is really a primitive kind of parallelization and because both are inhibited by many of the same types of data dependencies.

The relative performance of column oriented versus row oriented code is discussed in [10]. Dongarra, Gustavson, and Karp [4] have demonstrated that loop reordering gives vector or supervector performance for standard linear algebra loops on a Cray 1 type column oriented FORTRAN environment with vector registers. However, for the stochastic dynamic programming application, the dominant loops are non-standard linear algebra loops, so that the preference for column oriented loops is not a rule, as demonstrated on the Alliant vector multiprocessor [10].

Current efforts are concentrated on implementing the code on the Cray X-MP/48 and Cray 2 for more general multi-state and multi-control applications. In order to implement the code for arbitrary state space dimension, a more flexible data structure is needed for the problem arrays, $\mathbf{F}$, G and H, as well as for the solution arrays, V along with its derivatives and the control $\mathbf{U}$. In the straight-forward, original data structure, an array like the nonlinearity vector requires one index, $js(is)$, to denote a numerical node for each state variable is:

$$\mathsf{F(is,js(1),js(2),\cdots,js(m))} \tag{2.7}$$

for each state equation, $is = 1$ to m. It is assumed that there are a common number $M = M_1 = \cdots = M_m$ of nodes per state, so that $js(is) = 1$ to M for $is = 1$ to m states. As a consequence, the typically dominant loops containing the nonlinearity function $\mathbf{F}$, the solution gradient DV or similarly sized array are nested to a depth of at least $m + 1$. A typical loop has the form

```
      do 1 i = 1, m
         do 1 j1 = 1, M
                 :
            do 1 jm = 1, M
                    :
1                F(i,j1,j2,···,jm) = ·····
```

This state size dependent loop nest depth level of $m + 1$ inhibits the development of general multibody algorithms, especially when the state size m is incremented and the number of loops in each nest have to be changed. Also, vectorization is inhibited for compilers that vectorize only the most inner loop. Parallel and vector optimization is important, due to the size of the work load, which is $\mathcal{O}(m \cdot M^m)$, for the dominant loop illustrated above. As the

number of states grows the computational load will grow like some multiple of

$$m \cdot M^m = m \cdot e^{m \ln(M)},$$

i.e., the load grows exponentially in the number of states m. This exponential growth is merely a quantitative expression of *Bellman's curse of dimensionality.*

One way around this inhibiting structure (2.7) is to use a vector data structure:

$$\mathsf{FV(is,jv)} \tag{2.8}$$

for the nonlinearity vector as an example, such that all the numerical nodes are collected into a single vector indexed by the global state index jv, where $jv = 1$ to M^m over all state nodes. Assuming that the number of nodes per state are fixed at M, then for a fixed set of state node indices $\{js(1), js(2), \cdots, js(m)\}$, the global state vector index is computed from the direct mapping formula

$$jv = \sum_{i=1}^{m} (js(i) - 1) \cdot M^{i-1} + 1, \tag{2.9}$$

in the case of fixed state mesh size, $M_i = M$ for all states i.

Both the direct mapping from the original data structure to the vector data structure and the inverse mapping are needed to compute the amplitude functions, $\mathbf{F}$, G and H, as well as the derivatives of V^*, because these quantities depend on the original formulation. The pseudo-inverse of the vector index in (2.9) can be shown to permit the recovery of the individual state indices by way of integer arithmetic:

$$js(is; jv) = 1 + [jv - 1 - \sum_{i=is+1}^{m} (js(i; jv) - 1) \cdot N^{i-1}]/N^{is-1}, \tag{2.10}$$

recursively, for $is = m$ to 1, by back substitution, with $\sum_{i=m+1}^{m} a_i \equiv 0$. The vector data structure of (2.8) to (2.10) results in major do loop nests of the order of 1 to 2, rather than order of $m + 1$. A typical vector data structure loop has the form

```
      do  2  i = 1, m     ! parallel loop.
         do  2  jv = 1, M**m     ! vector loop.
                  ⋮
2                FV(i,jv) = ·····
```

resulting in a reduction of the loop nest depth from $m + 1$ to 2, independent of the number of states m. Preliminary implementation of the vector data structure is available on the Alliant multiprocessor and on the Cray X-MP/48.

One major disadvantage of the vector data structure given in (2.10) is that the largest degree of parallelism available to a parallel processor or multiprocessor in the most outer or state number loop is m, the number of states. This task load can be better scheduled on parallel processors by block decomposition or strip mining of the vector data structure loop in the index iv, so that the single inner loop is split into two evenly balanced loops (cf., Polychronopoulos [14]). Thus, dividing the vector data structure into blocks can enhance

parallelism. Let MBLK be the number of state nodes in each block and then the total number of blocks will be

$$\text{NBLK} = M^m/\text{MBLK},$$

assumed to be an integer for simplicity. Consequently, the blocked version of the typically dominant loop will have the form

```
      do 3 i = 1, m
         do 3 jblk = 1, MBLK    ! parallel loop.
            jv1 = 1 + MBLK*(jblk - 1)
            jv2 = MBLK*jblk
            do 3 jv = jv1, jv2    ! vector loop.
                  ⋮
3                FV(i,jv) = ·····
```

This form should result in better parallel optimization when there are more than m available parallel processors.

The advantages of the algorithm is that it 1) permits the treatment of general continuous time Markov noise or deterministic problems without noise in the same code, 2) maintains feedback control, 3) permits the cheap control limit to linear singular control to be found from the same quadratic cost code, 4) stable mesh selection can be used to control the number of corrector steps, and 5) produces very vectorizable and parallelizable code whose performance is described in the next section.

3. Results and Discussion. The stochastic dynamic programming code arose from renewable resource modeling problems of Hanson and co-worker Ryan, with various one-state, one-control models treated in [15] and [11]. Two-state, two-control models were treated by Hanson [8]. In the two-state model [8], the two controls represent removals from the system by respective commercial and recreational users of the system. Poisson noise is used to represent natural catastrophic events. Applications to aerospace problems only entails modification of the dynamical system and performance criteria input by appropriate aerospace input functions and parameters.

The dynamic programming code has been optimized for parallelization and vectorization [9] using Hanson's two-state model [8] as a test example, and the Alliant FX/8 vector multiprocessor as the advanced hardware. The Alliant FX/8 at the Advanced Computing Research Facility (ACRF) at Argonne National Laboratory was used for benchmarking the code. This Alliant FX/8 has eight vector computing elements (*CEs*). Each of the *CEs* has eight vector registers whose length is 32 eight-byte elements, and the *CEs* are connected to a 128 KB cache. Some automatic parallelization and vectorization is performed, but significant increases are still attainable by the removal of optimization hindering data dependencies. Benchmark performance was measured for many mesh sizes and on all processor configurations. Almost all loops were of the highly optimized parallel and vector type for the Alliant. Over 65% efficiency was achieved over a wide range of tests [9]. The temporal mesh was chosen to be about four times more refined than the spatial mesh, $K = 4 \cdot (M-1) + 1$, for a fixed number of spatial nodes M and for constant numerical stability conditions. In

addition, vector stride effects (resonance effects related to multiples of the vector register length of 32 on the FX/8) were found with non-standard performance in both column and row referencing environments [10].

The present results have been obtained for a three-state, three-control modification of Hanson's two-state resource model [8] and by implementing the vector data structure mentioned above. The present application contains a new interacting state with competition. The present code is in a form where it is much more convenient to change the application, the advanced computer intrinsics, and the number of states.

Table 1 compares the performance of the code on the ACRF Alliant FX/8 vector multiprocessor at Argonne National Laboratory, the NCSA Cray X-MP/48 vector supercomputer at Urbana, and the University of Illinois at Chicago IBM3081K as a scalar uniprocessor reference. The Cray X-MP/48 is a four processor pipelined vector multiprocessor, but the use of the X-MP is much more costly to use in parallel than the Alliant and so only single processor results are reported here for the X-MP. The Cray executing on one vector processor outperforms the Alliant using either one vector processor or the full eight vector processors, due to the more powerful pipelined processing unit on the Cray. The advantages of block decomposition with MBLK = 32 for eight Alliant processors are illustrated in the table, where the eight processor time has been reduced from about 52 to 33 seconds when $M = 16$, while the one processor time has increased dramatically for the block method. The IBM3081K scalar uniprocessor is much slower when $M = 8$ unblocked spatial nodes than any of the Alliant or Cray values at $M = 8$. However, as the spatial mesh size is refined to $M = 16$ spatial points, with a corresponding increase in work load, the IBM3081K performs between the one and eight processor Alliant, but still significantly below the CRAY performance.

Table 1: Comparative Performance of IBM 3081K, Alliant and Cray, for three state model.

Nodes		Method	IBM 3081K vs fortran, opt(3)	Alliant FX/8 fortran -O		Cray X-MP cft77
state M	time K		$p = 1$	$p = 1$	$p = 8$	$p = 1$
8	29	unblocked	38.513	8.653	2.980	0.144
16	61	unblocked	85.377	147.391	51.619	2.058
8	29	blocked	—	13.693	1.998	—
16	61	blocked	—	223.426	32.729	—

The performance of the stochastic programming code under parallel and vector operation is investigated in more detail on the ACRF Alliant FX/8, which has better parallel capability than the Cray X-MP/48. The Cray X-MP/48 is also a vector multiprocessor, but the multiprocessing features are not as easily accessed as on the Alliant, where parallelization is more transparent. In Figure 3, the blocked and unblocked code is compared on the Alliant FX/8 with time $T(p)$ plotted against the number of processors p. The unblocked code runs faster as the number of processors increases from one, but then ceases to run any

faster beyond $p = 3$ processors due to the fact that the maximum parallelism available is the three iterations in the three-state outer loop. The blocked code, using a block size of MBLK = 32 (the vector register length on the Alliant) runs faster the more processors used out of the eight vector processors. However, the unblocked code is faster for $p < 5$, but slower for $p > 5$. The trade-off point between the blocked and unblocked code is $p = 5$, with the block overhead slowing down the code for $p < 5$, but the benefit of parallelism is found for $p > 5$.

Figure 4 shows the speedup, $S(p) = T(1)/T(p)$, versus the number of processors p. The unblocked code clearly exhibits a speedup plateau for $p \geq 3$ and the unblocked code exhibits nearly ideal speedup, $S(p) \simeq p$ for all p. However, this figure illustrates the danger of comparing speedups, because the unblocked case is better for $p < 5$ as demonstrated in Figure 3. In Figure 5, the efficiency, $E(p) = S(p)/p$ or speedup per processor, versus the number of processors p is shown. Again, the blocked efficiency is much higher than the unblocked efficiency, independent of the actual performance.

4. CONCLUSIONS. Stochastic dynamic programming algorithm can be optimized to permit numerical solution of larger state space problems using vector multiprocessors. In order to handle a large number of state variables, a large number of parallel processors would be desirable, but Bellman's *curse of dimensionality* appears to very much weakened. Parallelization, vectorization, and general supercomputing are important in the solution of the larger problems. Robust mesh selection techniques are necessary to achieve stable algorithms. These techniques are generally applicable to other vector and parallel computers. The general code is valid for general Markov noise in continuous time, feedback control, nonlinear dynamics, nonlinear control and the cheap control limit.

Future directions include applications to aerospace problems, improved development of general code for an arbitrary number of state variables, enhanced code portability, extensions to Kalman filtering for imperfect observations, and optimization for other advanced architectures.

Acknowledgements. Work was supported, in part, by several Faculty Research Participantships, a Faculty Research Leave at Argonne National Laboratory Advanced Computing Research Facility, and by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U. S. Department of Energy, Under Contracts W-31-109-38 and DE-AC05-84-R21400; the National Science Foundation Grant DMS-8806099; the National Center for Supercomputing Applications in Urbana. The author wishes to acknowledge the work of C.-W. Leung for the advanced analysis of the vector data structure and for investigations of applications of Cray multitasking to the problem, and to S.-L. Chung for continued development of optimizations for the algorithm.

REFERENCES

[1] Alliant, **FX/FORTRAN Programmer's Handbook**, Alliant Computer Systems Corporation, Acton, Mass., 1985.

[2] M. Athans, D. Castanon, K. P. Dunn, C. S. Greene, W. H. Lee, N. R. Sandell, Jr.,

and A. S. Willsky, *The stochastic control of the F-8C aircraft using a multiple model adaptive control (MMAC) method - Part I: Equilibrium flight*, **IEEE Trans. Autom. Control**, vol. AC-22, pp. 768-780, 1977.

[3] R. E. Bellman, **Adaptive Control Processes: A Guided Tour**. Princeton: Princeton University Press, 1961.

[4] J. J. Dongarra, F. G. Gustavson, and A. Karp, *Implementation of linear algebra algorithms of dense matrices on a vector pipeline machine*, **SIAM Rev.**, vol. 26, pp. 91-112, 1984.

[5] J. Douglas, Jr., and T. DuPont, *Galerkin methods for parabolic equations*, **SIAM J. Num. Anal.**, vol. 7, pp. 575-626, 1970.

[6] **Future Directions in Control Theory: A Mathematical Perspective**, W. H. Fleming, Chairman. Philadelphia: Society for Industrial and Applied Mathematics, 1988.

[7] I. I. Gihman and A. V. Skorohod, **Controlled Stochastic Processes**. New York: Springer-Verlag, 1979.

[8] F. B. Hanson, *Bioeconomic model of the Lake Michigan alewife fishery*, **Can. J. Fish. Aquat. Sci.**, vol. 44, Suppl. II, pp. 298-305, 1987.

[9] F. B. Hanson, *Computational dynamic programming for stochastic optimal control on a vector multiprocessor*, **Argonne National Laboratory, Mathematics and Computer Science Division Technical Memorandum ANL/MCS-TM-113**, June 1988, 26 pages.

[10] F. B. Hanson, *Parallel computation for stochastic dynamic programming: Row versus column code orientation*, in **Proceedings 1988 Conference on Parallel Processing, Vol. III Algorithms and Applications**, D. H. Bailey, Editor. University Park: Pennsylvania State University Press, 1988, pp. 117-119.

[11] F. Hanson and D. Ryan, *Optimal harvesting with density dependent random effects*, **Natural Resource Modeling**, vol. 2, No. 3, pp. 439-455, 1988.

[12] D. Ludwig, *Optimal harvesting of a randomly fluctuating resource I: Application of perturbation methods*, **SIAM J. Appl. Math.**, vol. 37, pp. 166-184, 1979.

[13] K. Naimipour and F. B. Hanson, *Convergence of a numerical method for the Bellman equation of stochastic optimal control with quadratic costs*, In Preparation, 1989.

[14] C. D. Polychronopoulos, *Parallel Programming and Compilers*. Boston: Kluwer Academic Publishers, 1988, pp. 26-27.

[15] D. Ryan and F. B. Hanson, *Optimal harvesting of a logistic population in an environment with stochastic jumps*, **J. Math. Biol.**, vol. 24, pp. 259-277, 1986.

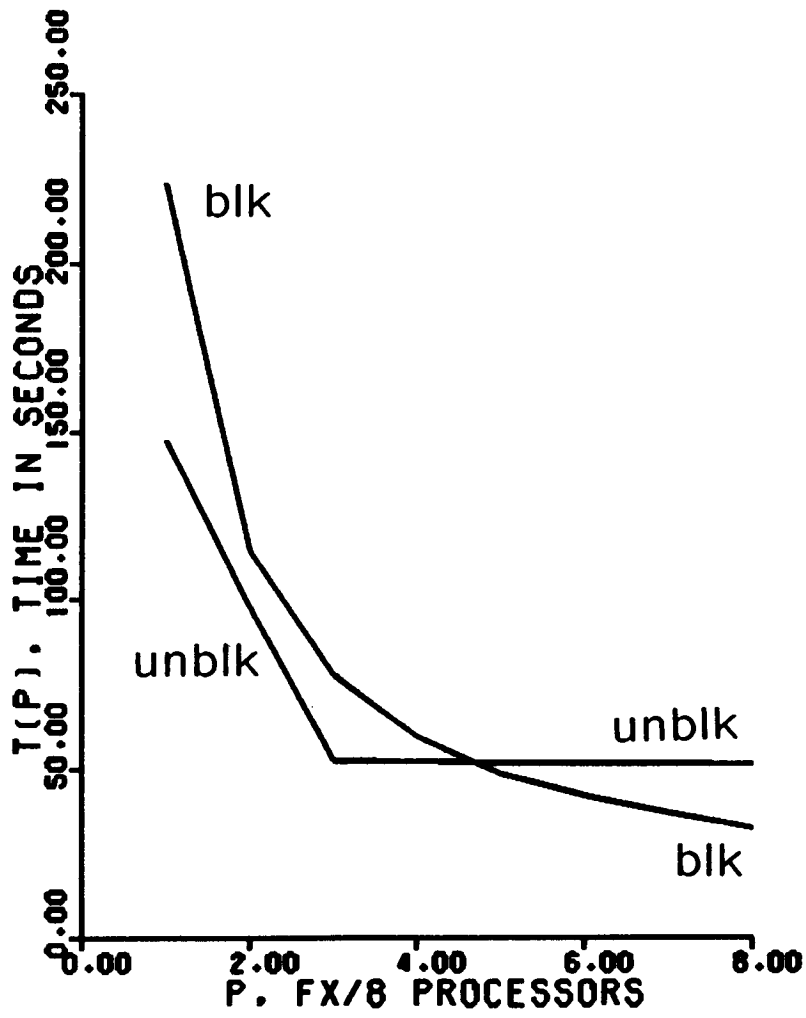

Figure 3: Comparison of blocked (blk) and unblocked (unblk) versions of the code. Time $T(p)$ is in seconds and p is the number of processors. Results are for $m = 3$ states, $M = 16$ spatial nodes and $K = 61$ temporal nodes.

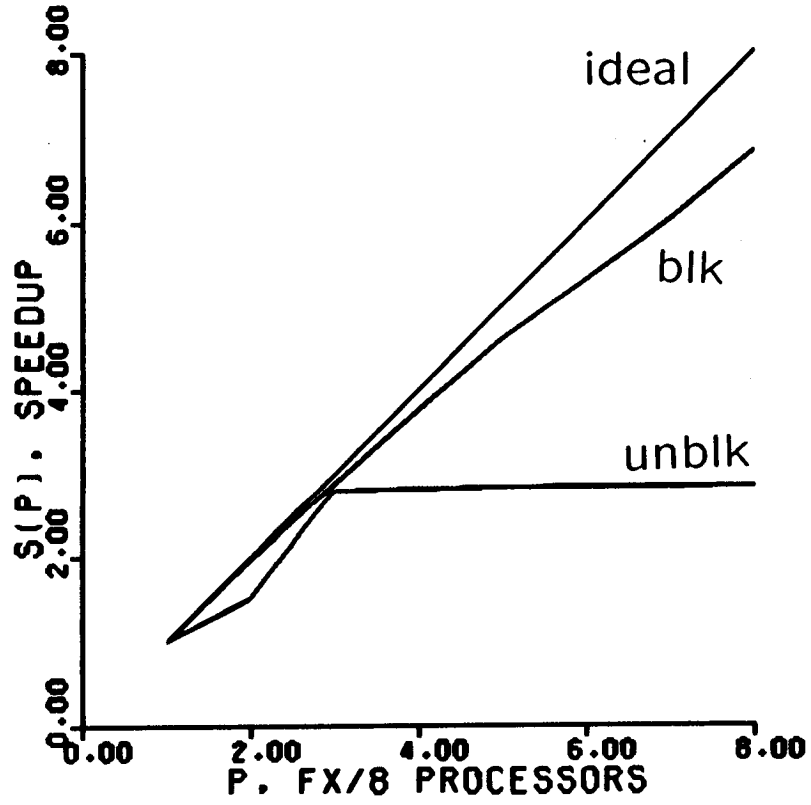

Figure 4: Speedups for blocked (blk) and unblocked (unblk) versions of the code. Speedup is denoted by $S(p) = T(1)/T(p)$ and p is the number of processors. The notation (ideal) denotes the ideal case, $S(p) = p$. Results are for $m = 3$ states, $M = 16$ spatial nodes and $K = 61$ temporal nodes.

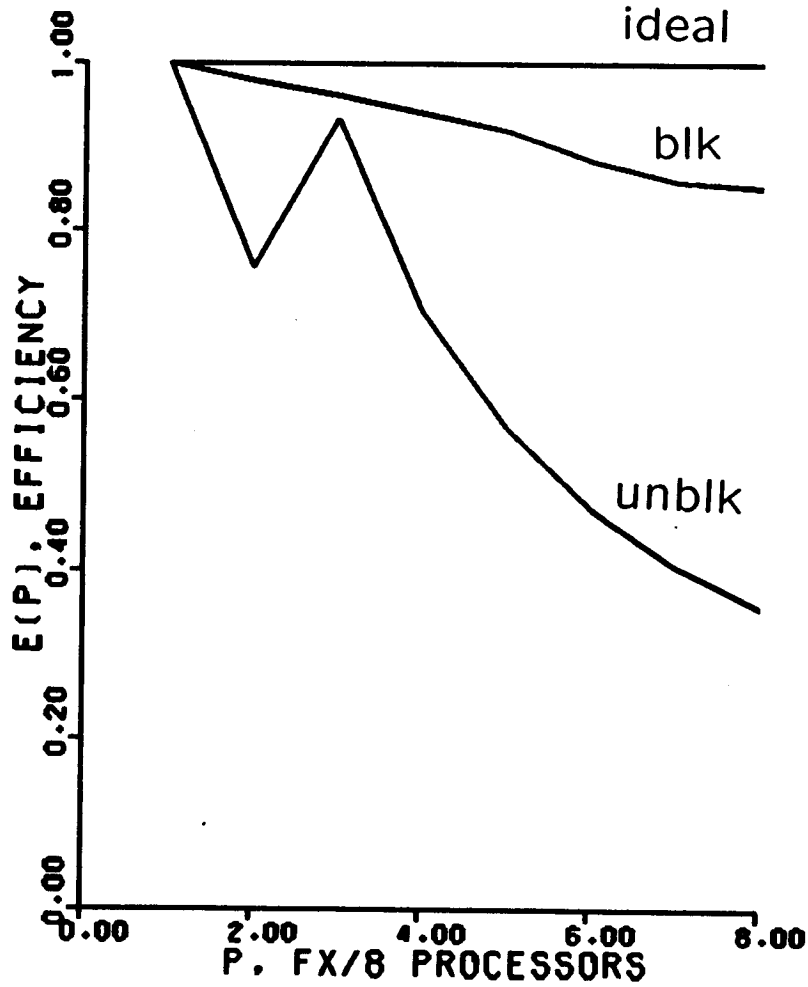

Figure 5: Efficiency for blocked (blk) and unblocked (unblk) versions of the code. Efficiency is denoted by $E(p) = S(p)/p$ and p is the number of processors. The notation (ideal) denotes the ideal case, $E(p) = 1$. Results are for $m = 3$ states, $M = 16$ spatial nodes and $K = 61$ temporal nodes.