Parallel Synthesis of Robust Control Systems

Wolfgang M. Schubert and Robert F. Stengel, *Fellow, IEEE*

***Abstract*—Parallel computing is used to evaluate compensator robustness numerically and to automate the design of control systems. Robustness is quantified by Monte Carlo evaluation of the effects of plant-parameter uncertainty, and it is maximized by searching the control-design-parameter space with a genetic algorithm. This design approach requires considerable computation, and parallel computing can reduce execution times. Because evaluation times vary among trials, computation time could be prolonged if some nodes remain idle while others finish their tasks. A dynamic scheduler is proposed as a solution. Theoretical and experimental results for a shared-virtual-memory computer illustrate that "speed-up" of controller design computations is nearly linear in the number of computing nodes.**

***Index Terms*— Control systems, design methodology, parallel processing.**

I. Introduction

PRACTICAL compensators must control plants satisfactorily in the presence of plant parameter uncertainties: that is, they must be *robust*. Robustness is best expressed as the probability that stability and performance of the controlled system fall within acceptable bounds of commonly used design metrics. For example, a specific degree of stability may be required, or limits may be imposed on settling time and actuator usage. In addition, it should be possible to verify robustness of plant operation in the presence of the most likely parameter variations by analysis or simulation.

Probabilistic analysis and design is a method for quantifying robustness and synthesizing a robust controller. It has been applied to various linear aircraft and robotic control problems [1]–[4], to a benchmark control problem posed at the 1990 American Control Conference [5]–[7], and to flight control of a nonlinear hypersonic aircraft [8]. There are two main components for analysis and design. *Monte Carlo evaluation* (MCE) provides estimates of robustness over the plant-parameter space in terms of probabilities of instability, settling-time violation, actuator saturation, and other performance metrics. A global search procedure such as the *genetic algorithm* (GA) searches over the control-design-parameter space. These numerical methods are called for because both the plant-parameter space and control-design-parameter space may have high dimension, producing a design problem that is computationally complex, whose deterministic solution may be intractable.

Probabilistic control system design is readily performed on personal computers or workstations, but the process can be time-consuming. For the closed-loop system designed in [7], satisfactory convergence required eight GA generations, with a total of about 70 000 MCE's. Running MATLAB code on a single-node, 16-MFlop workstation, a single design required several hours for computation. By comparison, a 512-node, 128-GFlop parallel processor would reduce execution times to the order of seconds. Such dramatic "speed-ups" will be needed to design complex control systems, such as those required for large aircraft, flexible robots, chemical processes, and large-scale structures. This paper explores the potential for solving intractable control design problems using parallel processing, taking into account hardware architecture and implementation issues.

II. Probabilistic Analysis and Design

A feasible controller that satisfies design constraints and minimizes a probabilistic cost function is defined by simulation and search [7]. A GA evolves a population of design-parameter strings (or "chromosomes") that are acted upon by four different operations: evaluation, selection, crossover, and mutation. After all four operators have been applied to each member of the population, a generation is complete. A feasible control system design emerges as the GA converges (in probability) to the global optimum in the design space D which contains all possible design-parameter strings [9].

Each chromosome encodes an instance of the parameter vector $\boldsymbol{d}$, denoting a point in D. The "fitness" of each member in the population is based on *evaluation* of an ensemble average over a random sampling of uncertain plant parameters at the point. In this case, the plant parameters are assigned values by random number generators, the probabilities of satisfying design metrics over the ensemble are estimated numerically, and high fitness is represented by low robustness cost (defined below). Elite *selection* retains the best member of the population, while tournament selection pairs off the remaining members and chooses the survivors (i.e., the fitter member from each pair). Clustering analysis creates "super-elite" members that are added to the next generation by interpolating across each cluster of chromosomes in the design space. *Crossover* splices two chromosomes that have been selected, generating two offspring that enter the next generation. *Mutation* allows the GA to visit all regions in D (in probability), avoiding premature convergence toward a local optimum. Both local and global versions of the mutation operator are applied. Optimal fitnesses from multiple runs form

Manuscript received August 12, 1996. Recommended by Associate Editor, E. G. Collins, Jr. This work was supported by the Federal Aviation Administration and the National Aeronautics and Space Administration under Grant 95-G-011, and used computers located at the Cornell Theory Center.

W. M. Schubert was with the Department of Mechanical and Aerospace Engineering, Princeton University, Princeton, NJ 08544 USA. He is now with Oliver, Wyman & Co.

R. F. Stengel is with the Department of Mechanical and Aerospace Engineering, Princeton University, Princeton, NJ 0854 USA.

Publisher Item Identifier S 1063-6536(98)08034-8.

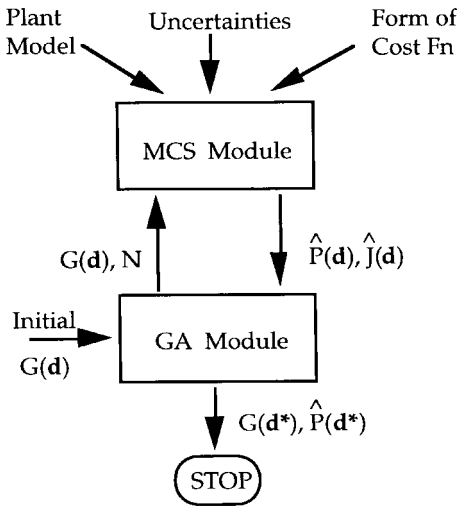

Fig. 1. The sequential probabilistic analysis and design algorithm.

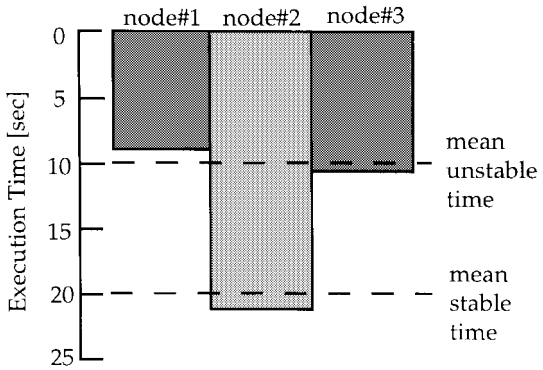

Fig. 2. Stochastic load imbalance during Monte Carlo evaluation.

a Weibull distribution [7]; hence, the global optimum can be estimated [10]. When the fittest member during a given run approaches this global optimum within a prescribed error, the design process terminates.

The sequential flow of control within the algorithm is depicted in Fig. 1. Let $G(\boldsymbol{d})$ be a pool of design-parameter chromosomes, initially created at random. The design cost function that judges the fitness of each chromosome for M design metrics is

$$J(\boldsymbol{d}) = \sum_{j=0}^{M} w_j P_j(\boldsymbol{d}) \tag{1}$$

where w_j is the weight assigned to the probability P_j that Design Metric j is violated, with P_j in [0, 1]. The design cost equals the weighted sum of all design metric probabilities. The forms of the metrics and their weights, as well as the plant model and the statistical characteristics of the uncertainties, are specified by the designer [1]–[8]. These earlier applications use P_j^2 rather than P_j in the design cost to diminish the effects of small probabilities and overweight the effects of high probabilities of violating design goals. The MCE module performs the evaluation, and the GA module performs selection, crossover, and mutation. In Fig. 1, $\hat{P}$ and $\hat{J}$ denote estimates computed by MCE. The accuracy of the estimates depends upon the number of MCE's N per chromosome and the actual probability [1]. As the best members in each generation approach the global optimum, the required number of evaluations increases to assure sufficient resolution in $\hat{P}$ during selection. The GA module computes the required N and passes it to the MCE module. At the completion of this iterative process, $G(\boldsymbol{d}^*)$ and $\hat{P}(\boldsymbol{d}^*)$ denote estimates of the best control design and its associated robustness probabilities.

The GA and the MCE modules of the sequential algorithm can be partitioned into a number of smaller subtasks. A chromosome encodes the design parameter $\boldsymbol{d}$, which, in this example, contains elements of linear-quadratic-Gaussian regulator (LQGR) weighting matrices. The LQGR requires two matrix Riccati equations to be solved for every member in the population. The GA could be applied with other design approaches (e.g., H_∞ or μ synthesis), or it could search for control gains directly. The computation is done independently for each chromosome, so the work can be distributed among multiple processors. The second parallelism lies in the MCE module. MCE's are performed for each of the controllers in the current population. These evaluations are independent from each other and can be distributed. After the MCE module has passed results back to the GA module, selection, crossover, and mutation are performed. This overall approach implies two synchronization points at which the software waits (or *blocks*) until all processors have completed their subtasks: the first after solving the Riccati equations and the second after performing all the MCE's.

Optimal speed-ups can be achieved only if all processors are busy continuously, that is, when the workload is *balanced*. Because of the logical flow structure within each MCE, different evaluations take different amounts of time to complete. For example, if an evaluation indicates unstable regulator eigenvalues, then none of the other design metrics (e.g., settling-time violation) need to be tested, and the MCE completes. The load balancing problem is stochastic because the amount of computation required at each trial is not known ahead of time. Fig. 2 illustrates the phenomenon for three nodes. Nodes 1 and 3 perform MCE's that predominantly indicate instability, whereas the Monte Carlo Evaluations performed on Node Two indicate more trials that are stable.

The stochastic load imbalance is examined for the design solutions of [7]. The three probabilities included in the cost function J are as follows.

P_i: *Probability of Instability*, portraying the likelihood that at least one of the closed-loop roots lies in the right-half plane, due to parameter variations.

P_{Ts}: *Probability of Settling-Time Exceedance*, estimating the likelihood that the absolute magnitude of the measurement falls outside a 0.1 unit envelope after 15 s.

P_u: *Probability of Actuator Saturation*, corresponding to the probability that the absolute magnitude of peak actuator displacement exceeds one unit.

The logical flow structure of one MCE is as follows. First the eigenvalues of the closed-loop system are computed. If the closed-loop system is stable, a unit-impulse disturbance input is propagated to test for settling-time exceedance and

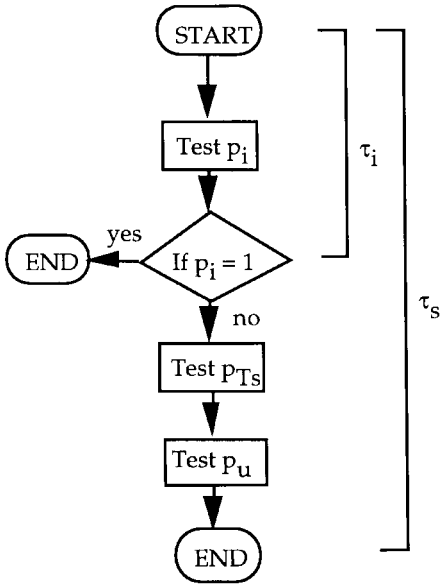

Fig. 3. Flow diagram for a single Monte Carlo evaluation.

actuator saturation. The flow is summarized in Fig. 3, which also indicates the two possible execution times for a single evaluation, τ_s and τ_i. The sample probabilities p_i, p_{Ts}, and p_u are set to zero or one depending on the outcome of the test, and the probabilities P_i, P_{Ts}, and P_u are the sums of the individual sample probabilities over the N trials.

Recalling that N is the number of MCE's to be performed per chromosome, we define L as the total number of processors available and p_s as the fraction of stable evaluations for an individual node (i.e., number of evaluations indicating stability divided by number of evaluations performed for a single node). If the workload is distributed such that each node computes N/L evaluations, then the slowest node will determine the overall execution time, because of the second synchronization point. In this ideal model, the slowest node is the one with the largest probability of stability $p_s(=1-p_i)$ or $p_{s\max}$. The actual execution time becomes

$$T_a = \frac{N}{L}[p_{s\max}\tau_s + \tau_i(1-p_{s\max})]. \tag{2}$$

If each processor runs a single MCE and if the total probability of stability is greater than zero, then $p_{s\max}$ is equal to one. If one processor performs all the work, then $p_{s\max}$ is equal to P_s, the sum of individual sample probabilities of stability. $p_{s\max}$ is a function of the number of processors used, L, whose shape depends on the problem and is unknown beforehand. Nevertheless, we can estimate $p_{s\max}$ as an affine function of the distribution's standard deviation, σ

$$p_{s\max} = P_s + k\sigma \tag{3}$$

From the assumption that p_s follows a Bernoulli distribution [10], the standard deviation can be computed as

$$\sigma = \left[\frac{P_s - P_s^2}{N/L}\right]^{1/2}. \tag{4}$$

As long as L/N is less than one-tenth, i.e., more than ten evaluations are performed per node, the Bernoulli distribution

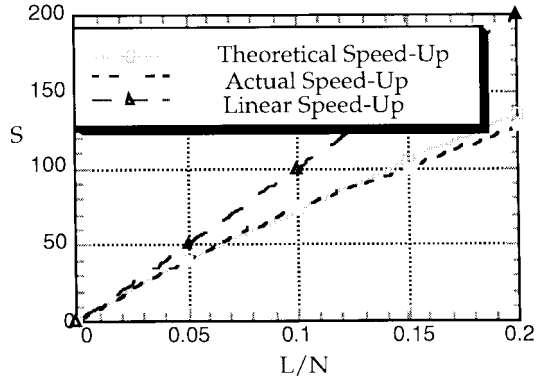

Fig. 4. Comparison of theoretical versus actual speed-ups. $N = 1000$, $s = 0.365$, $c = 2.3$, sequential simulation of parallel processing.

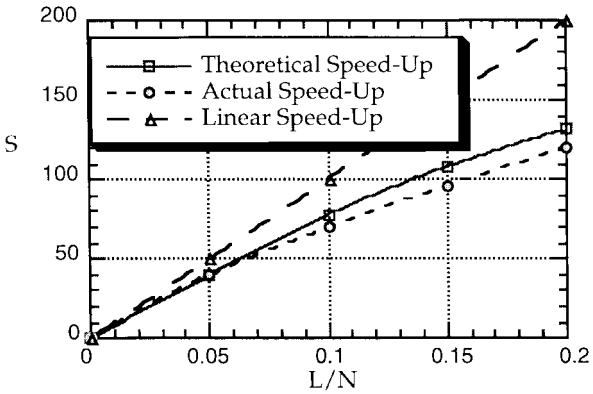

Fig. 5. Comparison of theoretical versus actual speed-ups. $N = 1000$, $s = 0.3$, $c = 2.3$, sequential simulation of parallel processing.

can be approximated by a normal distribution. For a normal distribution, the proportionality factor k is found from order statistics [11] to be

$$k = \sqrt{2\log L} - \frac{\log\log L + \log 4\pi}{2\sqrt{2\log L}} + \frac{\gamma}{\sqrt{2\log L}} \tag{5}$$

where γ is Euler's constant, ≈ 0.5772. Introducing the constant c as the ratio of τ_s/τ_i, the actual execution time ("wall-clock time") becomes

$$T_a = \frac{N}{L}[(P_s + k\sigma)(c-1) + 1]\tau_i. \tag{6}$$

The *speed-up ratio* or ratio of sequential execution time over actual execution time S is

$$S = L\frac{P_s(c-1)+1}{(P_s + k\sigma)(c-1)+1.} \tag{7}$$

S is a fraction of L as long as k and σ are greater than zero. Values for S are compared with experimental results from parallel-processing simulations (Figs. 4 and 5). The experiment was performed on a single-node workstation in order to avoid the contamination of the results by effects from interprocessor communication and other users that would occur on a parallel machine. For the simulation, packets of size N/L were passed to the MCE module L times in sequence. For each call to the MCE module, the number of floating-point operations was measured. The speed-up ratio

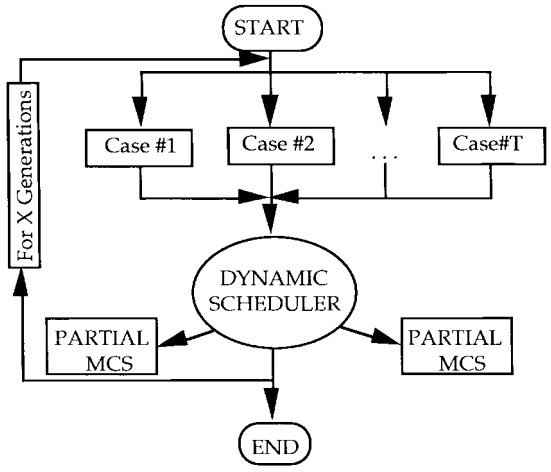

Fig. 6. The parallel stochastic robustness analysis and design algorithm.

S for the simulation was defined as the total number of operations performed divided by the maximum number of operations found.

The figures show the validity of (7) for $0 < L/N/, < 0.1$. For $L/N > 0.1$, the characteristics of the Bernoulli distribution do not allow an accurate estimate of $p_{s\max}$. For the two example runs shown, the efficiency, i.e., the ratio of actual speed-up over linear speed-up, is about 60%. This means that over one-third of the available computing power is wasted in idle cycles.

An asynchronous approach could be taken to avoid idle cycles. After a package size is determined, a *dynamic scheduler* would distribute MCE's to processors as they become available. Even if different packages take different amounts of time to compute, the load is optimally balanced. A dynamic scheduler is not necessary for the GA module, because the solutions to different Riccati equations take similar amounts of time (Fig. 6). The algorithm can be classified as being *two-phased* and *task-parallel*. The first phase includes the parallel computation of the controller gains, and the second phase covers the MCE's. Selection, crossover, and mutation are performed after completion of the second phase.

III. The Distributed Computing Environment

Not all parallel-processing architectures are well matched to Monte Carlo simulation for control design. Parallel simulation of device dynamics and control will benefit from coarse-grain parallel structures, and distributed memory is a greater concern than shared memory. Full connectivity, parallel message exchange, and short communication paths are helpful but not critical to efficient MCE in this application. A single-instruction/multiple-data-stream (SIMD) computer has a single control unit that sends identical instructions to multiple processing elements, and it depends on an interconnection network that has these properties; hence, SIMD computers are best applied to fine-grain problems like the solution of partial differential equations. If each processor were executing the same simulation (using different random parameters), N runs could be completed concurrently; however, local, parameter-dependent, logical branches would not be allowed.

TABLE I
Comparison of Two Parallel Computers

	KSR1	IBM SP1
Parallelism	Shared-Virtual	Message-Passing
Peak Performance per Processor (MFlops)	40	120
Local Processor Memory (MBytes)	32 (cache)	128
Maximum Memory per Process (GBytes)	1000	0.5

Even simple dynamic simulations typically contain logical branches that are parameter-dependent, so greater flexibility is needed. Multiple-instruction/multiple-data-stream (MIMD) computer architecture allows loose coupling and asynchronous operation of processors. A single-program/multiple-data-stream (SPMD) parallel computer retains the programming simplicity of the SIMD machine while allowing the greater independence of the MIMD unit; it would, therefore, be useful for Monte Carlo simulation.

Riccati equation solutions and MCE's are computationally intensive and imply a rather coarse-grained parallel approach. Little interprocessor communication is required during concurrent execution, which is more efficient than the larger amount of communication generated by more fine-grained approaches. *Monte Carlo logic is inherently scalable*, and it can use MIMD features implemented either as a unit or as a loosely connected network. Scalable software can readily be ported to computers with an arbitrary number of processors. As noted above, Monte Carlo simulation can be executed on one or more processors with linear speed-up; hence, it is maximally scalable.

Most parallel computers are characterized as either *distributed* (message-passing) or *shared-memory*, with a few hybrid exceptions. Machines with distributed physical memory generally scale better, i.e., their overall performance does not degrade as the number of processors increases. Probabilistic simulations could theoretically run on thousands of processors in parallel with efficiencies approaching linear speed-up. The Kendall Square Research KSR1 and IBM SP1 were examined in this study. The KSR1 uses a shared-virtual-memory communications protocol that is implemented on top of a distributed-physical-memory architecture [12]. Its memory is made up exclusively from local caches (32 MBytes each). Every cache is associated with a custom-made 40-MFlop peak-performance processor, and a maximum of 32 processors are tied together on a local slotted-ring. Several local rings can be connected via a ring of rings. The shared-virtual-memory protocol is convenient for writing code, but the nonuniform memory access costs resulting from the physical architecture complicate performance analysis. The IBM SP1 uses a message-passing communications protocol based on distributed physical memory [13]. The physical architecture uses 120-MFlop peak-performance processors, eight of which are associated with a *rack*. Several racks

can be connected via high-performance switches. Several message-passing paradigms are available, such as parallel virtual machine (PVM) [14], message passing interface (MPI) [15], and the shared-virtual-memory paradigm *Linda* [16]. Table I summarizes physical characteristics of the KSR1 and the SP1.

The software was coded using the C++ programming language and executed on computers located at the Cornell Theory Center. The object-oriented features of C++ are well suited to a modular development approach, and they allow for convenient code maintenance. The current version implements the MCE module without dynamic scheduling on the KSR1. The code attempts to minimize *memory latency* (i.e., delays in execution) problems by using intelligent data management techniques. Because of the KSR1's cache architecture, shared variables were *subpage-aligned*. This means that these variables do not get oddly distributed among a number of memory pages, which can lead to so-called *thrashing* where different processors alternately try to grab the same page. The KSR-specific *prefetch* instructions were used. A prefetch may be issued even before the specified data is actually accessed by the program, allowing the operating system to fetch the data in the background if it is located outside the cache. The program does not have to stand idle waiting for the data. Writes to shared variables were avoided, as these result in the locking of such variables, slowing down the software. Low-level parallelism was used throughout the code, relying on KSR's *POSIX threads* as the principle constructs for executing subtasks concurrently. Threads are sequential flows of control within a process that cooperate with each other to solve the overall problem. A program or process begins with one master thread that handles the creation, control, and destruction of additional worker threads. Each thread usually executes on a separate processor for the highest efficiency.

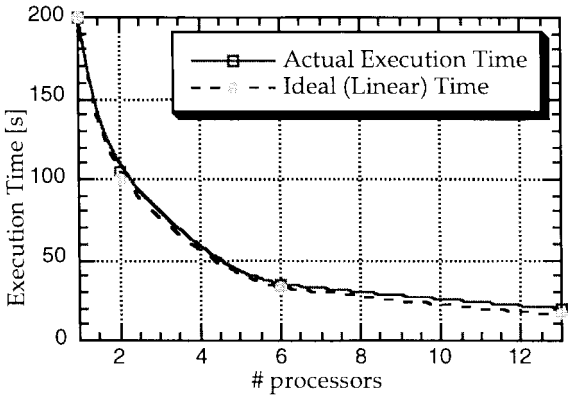

Fig. 7. Execution time of Monte Carlo evaluation module versus number of processors for KSR1 parallel computer.

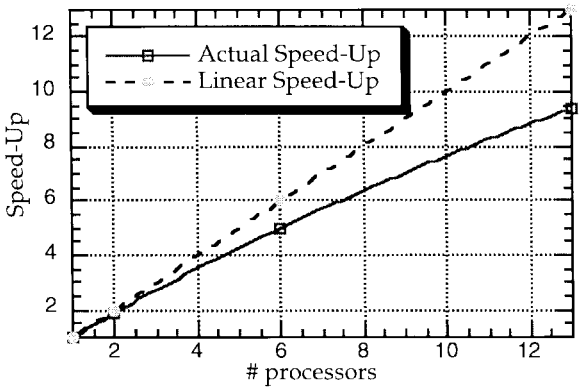

Fig. 8. Speed-up for Monte Carlo evaluation module versus number of processors for KSR1 parallel computer.

IV. Experimental Results

Performance results for the ACC benchmark problem have been produced for the MCE module without dynamic scheduling using the KSR1 and SP1 computers. The number of MCE's N was chosen to be 1000. Several runs were recorded, with one, two, six, and 13 processors (Fig. 7). The execution time was 200 s for the sequential run using only a single KSR1 processor. Similar MATLAB code running on a Silicon Graphics Indigo R4000 took 105 s. Fig. 7 shows that this speed is matched by using two processors on the KSR1. The minimum execution time for this sequence of runs was about 20 s using 13 processors, a five-fold speed-up when compared to the Indigo workstation.

Conversely, the speed-up ratio is plotted as a function of the number of processors in Fig. 8. The measured speed-up is almost linear for L less than four but exhibits a diminishing slope beyond that point. Using 13 processors, the measured speed-up is 73% of the linear speed-up. Because P_s is close to one in this example, we would expect an efficiency above 90%, even allowing for communication and system overhead. The discrepancy is due to several factors, including: suboptimal code that does not yet take full advantage of all available intelligent data management techniques, hardware limitations such as memory latency, and performance-degrading effects resulting from the operating system and the multiuser environment. Experiments using the SP1 computer produced similar results. For 13 processors, the theoretical speed-up is 92%, and the actual speed-up is 73%.

The results indicate that it is possible to achieve near-linear speed-ups in probabilistic analysis and design. As the design problems become harder, e.g., the system is of higher-order and the number of constraints and metrics increases, the total execution time would increase, but high computational efficiency would be retained.

V. Conclusions

A parallel-processing approach to robust control system analysis and design has been described, and preliminary results have been discussed. The method uses MCE and a GA, corresponding to the analysis and the design components. The work within each module is distributed and completed in parallel, with synchronization points between phases. Stochastic load imbalances affecting MCE have been analyzed, and a dynamic scheduler has been suggested to alleviate the problem. Initial experimental results indicate that near-linear speed-ups can

be achieved in practice, enhancing the utility of the design approach.

REFERENCES

[1] R. F. Stengel and L. R. Ray, "Stochastic robustness of linear-time-invariant control systems," *IEEE Trans. Automat. Contr.*, vol. 36, no. 1, pp. 82–87, Jan. 1991.

[2] ———, "Application of stochastic robustness to aircraft control systems," *J. Guidance, Contr., Dynamics*, vol. 14, no. 6, pp. 1251–1259, Nov.–Dec. 1991.

[3] ———, "Stochastic measures of performance robustness in aircraft control systems," *J. Guidance, Contr., Dynamics*, vol. 15, no. 6, pp. 1381–1387, Nov.–Dec. 1992.

[4] ———, "A Monte Carlo approach to the analysis of control system robustness," *Automatica*, vol. 29, no. 1, pp. 229–236, Jan. 1993.

[5] R. F. Stengel, L. R. Ray, and C. I. Marrison, "Probabilistic evaluation of control system robustness," *Int. J. Syst. Sci.*, vol. 26, no. 7, pp. 1363–1382, July 1995.

[6] C. I. Marrison and R. F. Stengel, "Stochastic robustness synthesis applied to a benchmark control problem," *Int. J. Robust Nonlinear Contr.*, vol. 5, no. 1, pp. 13–31, Jan. 1995.

[7] ———, "Robust control system design using random search and genetic algorithms," *IEEE Trans. Automat. Cont.*, vol. 42, pp. 835–839, June 1997.

[8] ———, "Design of robust control systems for a hypersonic aircraft," *J. Guidance, Contr., Dynamics*, vol. 21, no. 1, pp. 58–63, Jan.-Feb. 1998.

[9] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[10] C. I. Marrison, "The design of control laws for uncertain dynamic systems using stochastic robustness metrics," Ph.D. dissertation, Dept. Mech. Aerospace Eng., Princeton Univ., NJ, 1994.

[11] A. H.-S. Ang and W. H. Tang, *Probability Concepts in Engineering Planning and Design*. New York: Wiley, 1975.

[12] *KSR Parallel Programming* and *KSR C Programming*, Kendall Square Res. Corp., Waltham, MA, 1992.

[13] ———, *IBM Scalable POWER parallel Systems Reference Guide*, IBM, Armonk, NY, 1993.

[14] A. Geist and A. Beguelin, *et al.*, "PVM3 user's guide and reference manual," Oak Ridge Nat. Lab., Tech. Rep. ORNL/TM-12187, Oak Ridge, TN, May 1993.

[15] W. Gropp and E. Lusk, "A test implementation of the MPI draft message passing standard," Argonne Nat. Lab., ANL-92/47, Argonne, IL, Dec. 1992.

[16] A. H. Sherman, *C-Linda Ref. Manual*, Scientific Computing Associates, Inc., New Haven, CT, 1990.

Wolfgang M. Schubert received the B.S. degree from the University of Michigan, Ann Arbor, in 1993.

He is a Consultant with Oliver, Wyman, & Co., New York.

Robert F. Stengel (M'77–SM'83–F'93) received the S.B. degree from Massachusetts Institute of Technology, Cambridge, in 1960, and the M.S.E., M.A., and Ph.D. degrees from Princeton University, Princeton, NJ, in 1965, 1966, and 1968, respectively.

He was with The Analytic Sciences Corporation, Charles Stark Draper Laboratory, USAF, and NASA. He was principal designer of the Apollo Lunar Module manual control logic, and he contributed to Space Shuttle control system design. He is currently Professor of Mechanical and Aerospace Engineering and former Associate Dean of Engineering and Applied Science at Princeton University, where he has been a member of the faculty since 1977. He wrote *Optimal Control and Estimation* (New York: Dover, 1994), and he is the author of numerous technical papers. His research focuses on system dynamics, control, and machine intelligence.

Dr. Stengel is a Fellow of the AIAA and North American Editor of the Cambridge University Press Aerospace Series. Together with J. Hansman and R. Lilley, he received the first FAA Excellence in Aviation Award in 1997).