Fast-Floquet Analysis of Helicopter Trim and Stability With Distributed and Massively Parallel Computing*

S. Venkataratnam, Research Assistant
S. Subramanian, Research Associate
G. H. Gaonkar, Professor

Department of Mechanical Engineering
Florida Atlantic University
Boca Raton, FL 33431, USA

Abstract

Prediction of helicopter stability in trimmed flight is often based on Floquet theory. It involves a nonlinear trim analysis for the control inputs and corresponding periodic responses, and then a linearized stability analysis for the Floquet transition matrix (FTM), and eigenvalues and eigenvectors of this matrix. The shooting method with damped Newton iteration is used for the trim analysis and generates the FTM as a byproduct. The QR method is used for the eigenanalysis. The Floquet analysis comprises the shooting and QR methods. A rotor with Q identical and equally spaced blades has Q planes of symmetry. The fast-Floquet analysis exploits this symmetry, and thereby provides nearly a Q-fold reduction in run time and frequency indeterminacy of the Floquet analysis. Still, the run time for the fast-Floquet analysis on serial computers becomes prohibitive for large models (order or number of states > 100); in fact, it grows between quadratically and cubically with the order, and the bulk of it is for the trim analysis. Accordingly, a parallel fast-Floquet analysis is developed for both types of mainstream parallel computing hardware: distributed computing systems of networked workstations and massively parallel computers; algorithmically, both types belong to the MIMD (Multiple-Instruction, Multiple-Data) architecture. Large models with hundreds of states are treated, and a comprehensive database on parallel performance and computational reliability is generated. Computational reliability is quantified by parameters such as the eigenvalue condition number. Similarly, parallel performance is measured by parameters such as speedup and efficiency, which collectively provide a measure of how fast a job can be completed with a set of processors and how well the processors are utilized. Compared to the serial fast-Floquet analysis, the parallel analysis reduces the run time dramatically and provides a practical means of controlling the growth of run time with the order by a judicious combination of speedup and efficiency.

* *Paper presented at the 23rd European Rotorcraft Forum, Dresden, Germany, September 16–18, 1997.*

Nomenclature

Unless otherwise stated, the symbols below are nondimensional:

a	lift curve slope, rad^{-1}
C_{d_o}	constant profile drag coefficient
C_d	resultant profile drag force in the plane of the rotor disk opposite to the flight direction
C_l	rolling moment coefficient
C_m	pitching moment coefficient
C_T	thrust coefficient
C_w	weight coefficient of the helicopter
$\overline{f}$	equivalent flat plate area of parasite drag
f	serial fraction
M	number of states or state variables augmented with control inputs, $M = N + c$
N	number of structural and aerodynamic states or state variables
N_b	number of blade states
N_w	number of dynamic wake states
p	number of processors
P_β	flap natural frequency, rotating
Q	number of blades
t	time unit such that $T = 2\pi$
T	period
z_k	k-th eigenvalue of the FTM; see Eq. (2)
$\overline{z}_k$	k-th eigenvalue of EFTM; see Eq. (22)
α_j^r, β_j^r	wake states
ψ	azimuthal position
ξ_k	k-th mode nonunique frequency of z_k
$\overline{\xi}_k$	k-th mode nonunique frequency of $\overline{z}_k$
ΔT	T/Q
ϵ	perturbation quantity
γ	Lock number (blade inertia parameter)
σ	rotor solidity
σ_k	k-th mode damping of z_k
$\overline{\sigma}_k$	k-th mode damping of $\overline{z}_k$
μ	advance ratio
ω_ζ	lag natural frequency, rotating
$[\,]^T$	transpose of $[\,]$
$\|\,\|$	Euclidean norm of a vector or matrix
$(\dot{\mathrm{x}})$	time derivative of x

Introduction

Floquet theory is the primary mathematical tool to investigate helicopter stability in trimmed flight. The investigation involves two parts. The first part is the nonlinear trim analysis of computing the control inputs for the required flight conditions as well as the corresponding periodic or steady-flight response. The second part is the linearized stability analysis of generating the Floquet transition matrix (FTM) about the periodic response (Ref. 1), and then computing its eigenvalues and eigenvectors for the modal damping levels and frequencies. For the trim analysis, the shooting method with damped Newton iteration is used increasingly since it is not sensitive to the damping levels or stability margins, and therefore is suitable for marginally stable and unstable systems. Moreover, the shooting method is almost globally convergent and generates the FTM as a byproduct (Ref. 2). For the stability analysis, the QR method is used almost exclusively (Ref. 3). The trim analysis based on the shooting method and the stability analysis based on the QR method are collectively referred to as the Floquet analysis (Ref. 1). Moreover, the Floquet analysis has well-quantified computational reliability characteristics (Refs. 2–4) and it is routinely applied to small models on conventional serial computers (order or number of states $N < 100$). However, the run time for the trim analysis becomes prohibitive for large models; indeed, it grows between quadratically and cubically with the model order or number of states. Such a formidable run-time growth precludes the application of Floquet analysis to comprehensive and design analyses that require large models. Furthermore, as we exploit major strides in structural and aerodynamic modeling and thereby improve prediction capability, the required run time becomes prohibitive as well. Simply stated, the practical utility of the Floquet analysis becomes limited to relatively small models.

In the Floquet analysis, the trim analysis consumes nearly 99% of the total run time. It involves a simultaneous solution of nonlinear differential equations of motion coupled with algebraic transcendental equations of trim. Basically, the trim analysis predicts the control inputs that satisfy the required flight conditions and the corresponding initial conditions that guarantee periodic responses. These control inputs appear not only in the stiffness and damping matrices but also in the input matrix or forcing-function matrix. Therefore, they are specified indirectly so as to satisfy the prescribed flight conditions of required thrust level and force-moment equilibrium. Thus, the trim analysis is carried out iteratively by solving a sequence of linearized problems. This involves varying or perturbing the starting or initial values of the state variables and control inputs *one at a time*, integrating the equations of motion through *one complete period* T, generating the Jacobian and then improving the estimates by a damped Newton iteration. The cycle of perturbing, integrating and improving continues till convergence. Thus, the trim analysis requires a very large number of operations of perturbation, integration and improvement. To be precise, for an N-order model with c number of control inputs and k number of iteration cycles for convergence, the trim analysis requires $k(N + c + 1)$ operations, of which $(N + c + 1)$ operations in each iteration cycle are independent. It turns out that the time interval for all of the $k(N + c + 1)$ integration operations can be reduced from T to T/Q by exploiting the Q-planes of symmetry of a Q-bladed rotor; the only condition is that all the blades be identical and equally spaced. The conventional Floquet analysis that exploits this symmetry is referred to as the fast-Floquet analysis. Since the bulk of the run time is for such a large number of repeated integrations, the fast Floquet analysis reduces the run time of the Floquet analysis by nearly a factor of Q. It also turns out that the fast-Floquet analysis reduces the frequency indeterminacy of the Floquet analysis by a factor Q as well; for details see Refs. 5 and 6. Despite this Q-fold reduction, the run time for the trim analysis still grows rapidly, between quadratically and cubically with the number of states; this precludes treatment of large models even by the fast-Floquet analysis. Nevertheless, the $(N + c + 1)$ independent operations in each cycle can be tailored to parallel or concurrent computations. In principle, parallel computing reduces the run time by a factor of $(N + c + 1)$. Significantly, it also provides a means of controlling the run-time growth with the order by judiciously choosing the number of processors with the model size or number of states as a compromise between the run-time reduction and effective processor utilization. The fast Floquet analysis based on parallel computing is referred to as the parallel fast-Floquet analysis; it reduces the run time dramatically — theoretically, by a factor as large as $Q(N+c+1)$. To put this work in perspective, we present the state of the art of the conventional- and fast-Floquet analyses in the sequel.

A few studies are available on the serial and parallel Floquet analyses of large helicopter models (Refs. 7–9). For example, Ref. 7 addresses the application of the serial fast-Floquet analysis to models of order as high as 500. It also reviews the state of the art through 1995, particularly the basic studies of Peters (Ref. 5), and McVicar and Bradley (Ref. 6), and shows that the fast-Floquet analysis indeed brings in nearly a Q-fold saving in run time and reduces frequency indeterminacy by a factor of Q. However, the treatments therein show that the barrier of run time still remains for large models. Developments since 1995 include the works of Subramanian *et al.* who demonstrate that parallel computing is a means of removing such a

barrier (Refs. 8 and 9); in particular, Ref. 8 treats the parallel Floquet analysis, and Ref. 9, the parallel fast-Floquet analysis. However, these parallel analyses are designed for SIMD (Single-Instruction, Multiple Data) architecture and are not suitable for the MIMD (Multiple-Instruction, Multiple-Data) architecture that algorithmically forms the basis of both types of mainstream parallel-computing hardware — the massively parallel MIMD computers and distributed computing systems of networked workstations. More significantly, the MIMD architecture has been almost exclusively used in large-scale computing; indeed, the expectation is that except for some specialized applications (*e.g.*, weather prediction), the SIMD architecture will be virtually replaced by the MIMD architecture (details to follow).

Given this background, this paper advances the state of the art in the following respects:

1. It presents a parallel fast-Floquet analysis designed for distributed computing system of networked workstations and for massively parallel MIMD computers. The analysis exploits the fast-Floquet theory and the MIMD computational strategy.

2a. It treats models with hundreds of states to demonstrate the practical utility of the parallel fast-Floquet analysis in comprehensive- and design-analysis applications. The gains achieved through distributed and massively parallel computing are quantified in terms of the following parallel performance metrics: run-time and its growth with the order, the dominance of the parallel portion of the problem (serial and parallel fractions), speedup and efficiency. Descriptively stated, the speedup shows how the run time of a parallel algorithm running on p processors compares to itself when running on one processor. Similarly, efficiency provides a measure of how effectively the processors are used. Moreover, the computational reliability is quantified by the condition number of the Jacobian matrix in Newton iteration, the condition numbers of the eigenvalues and the residual errors of the eigenpairs.

2b. It presents a comprehensive database on parallel performance metrics and computational reliability. In particular, it shows how distributed computing compares with serial computing on a conventional main-frame computer and with massively parallel computing on a MIMD massively parallel computer with respect to performance and computational reliability. Since distributed computing is often built on existing hardware with little additional cost and with turnaround time of a workstation, this comparison demonstrates the practical utility of parallel computing in large-scale Floquet analysis.

Parallel Computing Hardware

In general, sequential algorithms are interchangeable; that is, the same algorithm can be implemented on all types of serial computers ranging from PCs to workstations to mainframe computers. By comparison, such a portability does not yet exist for parallel algorithms, which generally belongs to either SIMD or MIMD architecture. These two types of parallel architecture and their impact on the development of algorithms are extensively discussed in the literature (*e.g.*, Ref. 10). With respect to cost and speed, the MIMD computers are dominating the field of parallel computing. Basically, they utilize "off-the-shelf" processors, the same used in low-cost workstations, and exploit a chip technology that is making exponential growth in processing speed. In fact, the processing rate of an individual processor is expected to exceed 200 MFLOPS by the turn of the century (Ref. 11). From an algorithmic perspective, the distributed computing system belongs to the MIMD computing. This means when based on a same message-passing programming paradigm, the algorithms developed for a distributed computing system can be implemented directly on massively parallel MIMD computers and vice versa; this ensures portability of parallel algorithms. The concept of distributed computing is a relatively recent development, mostly since the early '90s. A brief account of it is included in the sequel, primarily to provide a better appreciation for the simplicity and utility of the parallel fast-Floquet analysis based on distributed computing. This account is intentionally descriptive with minimal use of parallel-computing jargon; for a thorough account see Ref. 11.

Distributed Computing

Distributed computing or network computing involves a set of networked computers ranging from a general-purpose workstation to a high-performance computer to even a parallel computer that works as a unified computing resource. In practice, however, a distributed computing system represents a cluster of heterogeneous workstations networked together, and its computational power can be comparable to that of a supercomputer. This networking in no way interferes with the stand-alone operation of individual workstations; indeed, one of the drivers of distributed computing is the feasibility of combining and realizing the unutilized computational power of these individual workstations during off-load (*e.g.*, after-office) hours. This unutilized computing power merits special consideration since a typical workstation routinely delivers several tens of millions of floating point operations per second and its computing power is expected to increase rapidly. Computing facilities from academia to design offices provide access to a large number of workstations through some form of interconnection network. The bulk of these workstations

remains idle for most of the day. Distributed computing provides a means of harnessing this untapped computing power. As far as a user's program is concerned, it acts as one single virtual parallel computer, no matter how many workstations are connected, how different they are and where they are located.

In contrast, a massively parallel computer represents an assemblage of a few hundred to a few thousand identical or homogeneous computers or processors and provides enormous computing power. Despite the architectural contrast — heterogeneous versus homogeneous processors — both the distributed computing systems and massively parallel computers use the same programming paradigm for interprocessor communication, which ensures all processors understand and exchange data. Although several libraries for interprocessor communication are developed by different vendors, the MPI (Message Passing Interface) in the public domain has evolved as a standard. The user need not worry about the architectural differences among the processors and related consequences such as distribution of tasks and data exchange. In other words, the MPI library spans the collection of heterogeneous processors in a distributed computing system as it does in a massively parallel computer with identical processors, although in the latter case this spanning is done much faster.

The increasing adaptation of distributed computing systems with massively parallel computers as necessary or highly desirable adjuncts is due to two factors. First, distributed computing not only requires minimum turnaround time but also involves very little cost since it is built on existing system hardware as well as software. Thus, it makes parallel computing practical, which is aptly referred to as "lowly parallel computing" (Ref. 12). By comparison, state-of-the-art massively parallel computers typically cost more than $10 million, and consequently they are maintained by only a few organizations. Since they are heavily used, the turnaround time often runs into days for a typical large-scale Floquet analysis ($N \approx 400$). Second, common to both types is the message passing programming paradigm (*e.g.*, MPI); this means the same algorithm and in fact, the same code developed for a distributed computing system can be run on a massively parallel computer as well. Thus, in the development of a prediction code, the bulk of the computations can be done on a distributed computing system at very little cost and turnaround time, and the 'final-stage' computations and other demonstration models can be run on a massively parallel computer. Stated otherwise, a judicious combination of distributed and massively parallel computing removes the run-time constraint that now prevents the routine application of Floquet theory to large models.

Conventional and Fast Floquet Analyses

For a rotor with Q identical and equally spaced blades, there are Q planes of symmetry. Exploiting this symmetry, the fundamental solution matrix or the transition matrix over one period T can be constructed from the transition matrix generated over T/Q or ΔT. The fast-Floquet analysis does just that. It exploits the Q-fold symmetry in both trim and stability analyses. Specifically, in the trim analysis, the equations of motion are integrated through ΔT and the trim equations are described in a Q-th part of a revolution, not through T or in one complete revolution as required in the conventional Floquet analysis. As a byproduct, the trim analysis also generates an equivalent Floquet transition matrix (EFTM). The eigenvalues and eigenvectors of the EFTM not only lead to stability results but also reduce the frequency indeterminacy by a factor of Q. Thus, in principle, it is possible to perform the Floquet analysis in $1/Q$-th of the run time for the conventional Floquet analysis. However, from an algorithmic perspective, the implementation of the fast-Floquet analysis requires considerable changes and these changes have significant impact on subsequent parallelization. Therefore, it is expedient to begin with a discussion of the stability as well as response of linear and nonlinear systems; this is followed by an outline of the shooting strategy based on the conventional Floquet analysis.

Conventional Floquet Analysis

Consider a linear periodic-coefficient system with N states, whose equations of motion are given by

$$\dot{\mathbf{x}}(t) = \mathbf{A}(t)\,\mathbf{x}(t) + \mathbf{G}(t) \tag{1}$$

where $\mathbf{x}(t)$ represents the $N \times 1$ state vector. Similarly, $\mathbf{A}(t)$ denotes the state matrix of size $N \times N$ and $\mathbf{G}(t)$ denotes the forcing function or input vector of size $N \times 1$; they are periodic with period $T = 2\pi$. For such a system, the $N \times N$ state transition matrix $\phi(t)$ is the fundamental solution matrix of the differential equation

$$\dot{\phi} = \mathbf{A}(t)\phi, \quad 0 \le t \le 2\pi, \quad \phi(0) = \mathbf{I} \tag{2}$$

and the FTM is defined by $\phi(2\pi)$. Then the stability of this linear system is determined by the eigenvalues z_k $(k = 1, 2, \ldots N)$ of the FTM, from which the modal damping levels and frequencies are computed:

$$\sigma_k = \frac{1}{2\pi} \ln |z_k| \tag{3a}$$

$$\xi_k = \frac{1}{2\pi} \arg(z_k) = \frac{1}{2\pi} \tan^{-1} \left(\frac{\mathrm{Im}(z_k)}{\mathrm{Re}(z_k)} \right) \tag{3b}$$

Furthermore, the initial conditions that guarantee periodic forced response for Eq. (1) are given by

$$[\mathbf{I} - \phi(2\pi)]\,(\mathbf{x}(0) - \mathbf{x}_E(0)) = (\mathbf{x}_E(2\pi) - \mathbf{x}_E(0)) \tag{4}$$

where $\mathbf{x}_E(2\pi)$ is the non-periodic solution at $t = 2\pi$ for any arbitrary initial state $\mathbf{x}_E(0)$.

In general, the governing equations of motion for rotorcraft systems are nonlinear periodic-coefficient differential equations and can be represented by

$$\dot{\mathbf{x}} = \mathbf{G}(\mathbf{x}, t) \tag{5}$$

For Eq. (5), the initial conditions that yield periodic forced response can be obtained by using a damped Newton-type iteration. In particular, for the k-th iteration, Eq. (4) is modified as

$$\begin{aligned} \mathbf{x}(0)_{k+1} &= \mathbf{x}_E(0)_k + \chi[\mathbf{I} - \phi(2\pi)]_k^{-1} \\ &\quad (\mathbf{x}_E(2\pi) - \mathbf{x}_E(0))_k \end{aligned} \tag{6}$$

where the matrix $[\mathbf{I} - \phi(2\pi)]$ is the Jacobian or partial derivative matrix Φ and χ is the Newton damping parameter. Moreover, $\phi(2\pi)$ converges to the FTM.

Showing the role of unknown control-input vector c explicitly, Eq. (5) can be recast as

$$\dot{\mathbf{x}} = \mathbf{G}(\mathbf{x}, \mathbf{c}, t) \tag{7}$$

Then, the initial state $\mathbf{x}(0)$ that generates periodic forced response and the control-input vector c that gives the desired flight conditions are computed by satisfying

$$\mathbf{x}(2\pi, \mathbf{x}(0)) - \mathbf{x}(0) = \mathbf{0} \tag{8}$$

$$\mathbf{f}(\mathbf{x}, \mathbf{c}) = \mathbf{0} \tag{9}$$

In the above equations, $\mathbf{x}(2\pi, \mathbf{x}(0))$ is the state at $t = 2\pi$ with initial condition $\mathbf{x}(0)$ and $\mathbf{f}(\mathbf{x}, \mathbf{c})$ symbolically represents the force-moment balance for the required flight conditions. Therefore, Eqs. (8) and (9) represent, respectively, periodicity and desired flight conditions and are nonlinear algebraic-transcendental equations. Therefore, combining Eqs. (8) and (9), we have

$$\mathbf{f}(\mathbf{s}) = \mathbf{0} \tag{10}$$

where $\mathbf{s} = \lfloor \mathbf{x},\ \mathbf{c} \rfloor^T$ denotes the augmented vector of state variables and control inputs.

Following Ref. (2), Eq. (6) is rewritten to include the computation of the control-input vector c as

$$\begin{aligned} \left\{ \begin{array}{c} \mathbf{x}(0) \\ \mathbf{c} \end{array} \right\}_{k+1} &= \left\{ \begin{array}{c} \mathbf{x}_E(0) \\ \mathbf{c} \end{array} \right\}_k \\ &\quad - \chi \left[\begin{array}{cc} \Phi_{11} - \mathbf{I} & \Phi_{12} \\ \Phi_{21} & \Phi_{22} \end{array} \right]^{-1} \\ &\quad \left\{ \begin{array}{c} \mathbf{x}_E(2\pi) - \mathbf{x}_E(0) \\ \delta \end{array} \right\} \end{aligned} \tag{11}$$

where δ represents the error in satisfying Eq. (9); that is, $\mathbf{f}(\mathbf{x}, \mathbf{c}) = \delta$. Moreover, the matrix Φ is the Jacobian and the submatrix Φ_{11} converges to the FTM.

Fast Floquet Analysis

In trimmed flight, variations in the blade sectional angle of attack and therefore, the air velocity components, rotor forces and moments are periodic. This means that the contribution of each identical blade to the rotor forces and moments will be the same at a given azimuthal position since each azimuthal position has its own blade pitch and corresponding air velocity. Therefore, the period of oscillations of rotor forces and moments is determined by the number of blades. Thus, it is sufficient to rotate through ΔT so that a Q-bladed rotor has a blade in all azimuthal positions instantaneously, and the variation of trim forces and moments in one period (T) can be described completely in ΔT. Thus,

$$\left\{ \begin{array}{c} C_T \\ C_d \\ C_l \\ C_m \end{array} \right\}_{\psi = 0} = \left\{ \begin{array}{c} C_T \\ C_d \\ C_l \\ C_m \end{array} \right\}_{\psi = \Delta T} \tag{12}$$

Since the blade sectional circulatory lift that influences the inflow forcing functions can also be described completely over the period ΔT, the periodicity for inflow or wake states can be established from

$$\left\{ \begin{array}{c} \alpha_j^r \\ \beta_j^r \end{array} \right\}_{\psi = 0} = \left\{ \begin{array}{c} \alpha_j^r \\ \beta_j^r \end{array} \right\}_{\psi = \Delta T} \tag{13}$$

However, for the blade states, it is required to rotate the rotor through a complete period T so that each blade passes through all azimuthal positions. (The blade states can include structural states such as displacements and velocities, and blade-fixed aerodynamic states such as stall states.) This means solutions over one complete period are required to establish the periodicity of the blade states in trim. Nevertheless, the interval for periodicity can be reduced to ΔT since all the blades follow the same trajectory as they go through a complete rotation but with a phase shift of ΔT between the paths of each blade. Hence, the states of an arbitrary q-th blade at an azimuthal position $\psi = \Delta T$ can be mapped onto the initial states of an identical $(q+1)$-th blade at $\psi = 0$. Therefore, the periodicity condition for the blade states becomes

$$\mathbf{x}_{b_{\psi = \Delta T}} = \mathbf{P}_b \mathbf{x}_{b_{\psi = 0}} \tag{14}$$

where $\mathbf{P}_b$ represents the $QN_b \times QN_b$ permutation matrix and $\mathbf{x}_b$ is the $QN_b \times 1$ vector of blade states, which is defined as

$$\mathbf{x}_b = \lfloor \mathbf{x}_{\text{blade 1}},\ \mathbf{x}_{\text{blade 2}},\ \cdots, \mathbf{x}_{\text{blade } Q} \rfloor^T \tag{15}$$

For an isolated rotor, $\mathbf{P}_b$ is given by

$$\mathbf{P}_b = \left[\begin{array}{cccccc} 0 & \mathbf{I}_b & 0 & \cdots & 0 & 0 \\ 0 & 0 & \mathbf{I}_b & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ldots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & \mathbf{I}_b \\ \mathbf{I}_b & 0 & 0 & \cdots & 0 & 0 \end{array} \right] \tag{16}$$

where $\mathbf{I}_b$ is the $N_b \times N_b$ unit matrix. However, Eq. (16) requires minor changes when body states are included; for details, see Ref. (5).

Now, combining the blade states and the inflow states, we write

$$\{\mathbf{x}\}_{\psi\ = \Delta T} = \mathbf{P}\,\{\mathbf{x}\}_{\psi\ = 0} \tag{17}$$

In the above equation, the $N \times N$ permutation matrix P is given by

$$\mathrm{P} = \begin{bmatrix} 0 & \mathrm{I}_b & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & \mathrm{I}_b & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & \mathrm{I}_b & 0 \\ \mathrm{I}_b & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & \mathrm{I}_w \end{bmatrix} \tag{18}$$

where I_w represents the unit matrix of size $N_w \times N_w$. From Eqs. (12–17), it is seen that the rotor trim forces, moments and periodic responses can be described in the interval ΔT. Accordingly, Eq. (11), which improves the initial conditions and the unknown control inputs is modified as (Ref. 6)

$$\begin{Bmatrix} \mathrm{x}(0) \\ \mathrm{c} \end{Bmatrix}_{k+1} = \begin{Bmatrix} \mathrm{x}_B(0) \\ \mathrm{c} \end{Bmatrix}_k - \chi \begin{bmatrix} \Phi_{11} - \mathrm{P} & \Phi_{12} \\ \Phi_{21} & \Phi_{22} \end{bmatrix}^{-1} \begin{Bmatrix} \mathrm{x}_B(\Delta T) - \mathrm{P}\mathrm{x}_B(0) \\ \delta \end{Bmatrix} \tag{19}$$

Furthermore, in Eq. (19), the matrix $\mathbf{P}^{\mathrm{T}}\mathbf{\Phi}_{11}$ converges to $[\mathrm{P}\ \phi(\Delta T)]$ where $\phi(\Delta T)$ is the transition matrix at the end of ΔT. The matrix $[\mathrm{P}\ \phi(\Delta T)]$ plays a major role in determining the stability, which is further elaborated in the sequel.

According to the fast-Floquet theory, the state transition matrix generated over the interval from $\psi = \Delta T$ to $\psi = 2\Delta T$ is identical to the one computed over the interval from $\psi = 0$ to $\psi = \Delta T$; the only requirement is that the blade indices be permuted. This is because at $\psi = \Delta T$ the first blade is in the same position as the second blade at $\psi = 0$. Therefore, the transition matrix between any two instants that differ by ΔT can be found from the transition matrix computed from $\psi = 0$ to $\psi = \Delta T$ and the permutation matrix P. The general relation between any two instants $\psi = n\Delta T$ to $\psi = (n+1)\Delta T$ is given by (Ref. 5):

$$\mathrm{P}^n \mathrm{x}[(n+1)\Delta T] = \phi(\Delta T)\mathrm{P}^n \mathrm{x}[n\Delta T] \qquad n = 0,\ 1,\ \ldots,\ Q-1 \tag{20}$$

From Eq.(20), we obtain

$$\mathrm{P}\mathrm{x}(\Delta T) = \mathrm{P}\phi(\Delta T)\mathrm{x}(0) \tag{21a}$$

$$\mathrm{P}^2\mathrm{x}(2\Delta T) = [\mathrm{P}\phi(\Delta T)]^2\,\mathrm{x}(0) \tag{21b}$$

$$\mathrm{P}^Q\mathrm{x}(Q\Delta T) = \mathrm{x}(2\pi) = [\mathrm{P}\phi(\Delta T)]^Q\,\mathrm{x}(0) \tag{21c}$$

which leads to

$$\phi(2\pi) = [\mathrm{P}\phi(\Delta T)]^Q \tag{22}$$

Thus, Eq. (22) relates the FTM $\phi(2\pi)$ and the matrix $[\mathrm{P}\phi(\Delta T)]$; see also Eq. (19). In other words, the FTM can be obtained by simply raising the power of $[\mathrm{P}\phi(\Delta T)]$ to Q. Similarly, the eigenvalues z_k of the FTM can be found from the eigenvalues $\overline{z}_k$ of $[\mathrm{P}\phi(\Delta T)]$ using the relation $z_k = \overline{z}_k^Q$. Since we are interested in the modal damping levels and frequencies, which are computed by taking the logarithm of z_k (see Eq. (3)), it is just sufficient to take the logarithm of $\overline{z}_k$ and multiply it by Q:

$$\overline{\sigma}_k = \frac{Q}{2\pi} \ln |\overline{z}_k| \tag{23a}$$

$$\overline{\xi}_k = \frac{Q}{2\pi} \arg(\overline{z}_k) = \frac{Q}{2\pi} \tan^{-1}\left(\frac{\mathrm{Im}(\overline{z}_k)}{\mathrm{Re}(\overline{z}_k)}\right) \tag{23b}$$

As seen from Eqs. (23b) and (3b), the frequencies are computed from an inverse arctangent function, which results in multiple values and merits additional comments. In the conventional Floquet analysis, the frequencies are unique up to the addition of $\pm j\Omega$, where $j = 0, 1, 2$, etc. By comparison, in the fast-Floquet analysis, the frequencies are unique up to the addition of $\pm j\Omega Q$; also see Eqs. (23b) and (3b). Thus, the fast-Floquet analysis reduces the frequency indeterminacy by a factor of Q. This is further corroborated by the numerical results generated from these two analyses in which the mode-identification method of Ref. 13 is used. Moreover, since the matrix $[\mathrm{P}\phi(\Delta T)]$ has one-to-one equivalence to the FTM, it is also referred to as the equivalent FTM (EFTM).

Parallel Fast-Floquet Analysis

Trim

An algorithm to trim an isolated rotor is presented; it is designed for MIMD computing systems, including distributed computing systems, and is based on the fast-Floquet theory and parallel shooting with damped Newton iteration. We consider a single-rotor model with N structural and aerodynamic states, and c number of control inputs. The bulk of the run time is for generating the $(N+c) \times (N+c)$ Jacobian in each iteration cycle. This involves repeated integrations of the equations of motion and estimation of trim forces, which are performed sequentially by the sequential shooting algorithm. In other words, the $(N+c)^2$ elements of the Jacobian are generated sequentially one element at a time. By comparison, the parallel algorithm generates these elements in parallel by suitably dividing the computations among the available processors. Specifically, it generates each column of the Jacobian by one processor. To help explain these significant features, we begin with a discussion of the sequential shooting algorithm.

Sequential Fast Shooting

The sequential shooting algorithm has the following seven instructions:

1. Assume M $(= N + c)$ arbitrary starting or initial values for the state variables of the augmented

state vector s; that is, N values for $\mathbf{x}(0)$ and c values for the control-input vector c.

2. Form the permutation matrix $\mathbf{P}$ of size $N \times N$ according to Eq. (18).

3. Perturb the M initial values *one value at a time* by a small amount ϵ_i, $i = 1, 2, \ldots, M$ and form $M + 1$ vectors of starting values:

$$\mathrm{s}+\begin{Bmatrix} \epsilon_1 \\ 0 \\ \vdots \\ 0 \end{Bmatrix}, \mathrm{s}+\begin{Bmatrix} 0 \\ \epsilon_2 \\ \vdots \\ 0 \end{Bmatrix}, \cdots, \mathrm{s}+\begin{Bmatrix} 0 \\ 0 \\ \vdots \\ \epsilon_M \end{Bmatrix}, \mathrm{s} \quad (24)$$

4. Integrate Eq. (7) for $M+1$ times using the $M+1$ vectors of starting values through a time interval $\Delta T = 2\pi/Q$ and generate the solution vectors:

$$\mathrm{y}^i = \begin{Bmatrix} \mathrm{x}(\Delta T) \\ \delta \end{Bmatrix}_{s+\epsilon_i} \text{ where } i = 1, 2, \ldots, M$$

$$\text{and } \mathrm{y} = \begin{Bmatrix} \mathrm{x}(\Delta T) \\ \delta \end{Bmatrix}_{s} \quad (25)$$

where the vector δ represents the trim error in satisfying Eq. (9). Moreover, the subscripts s and $s + \epsilon_i$, respectively, indicate the differences in the starting values; that is, one solution vector, y, with starting-value vector s and M solution vectors, y^i $(i = 1, 2, \cdots M)$ with M vectors of perturbed starting values.

5. Form the M columns of the Jacobian matrix Φ using

$$\left\{\frac{\mathrm{y}^i - \mathrm{y}}{\epsilon_i}\right\}, \quad i = 1, 2, \ldots, M \text{ or equivalently}$$

$$\Phi = \begin{bmatrix} \Phi_{11} - \mathrm{P} & \Phi_{12} \\ \Phi_{21} & \Phi_{22} \end{bmatrix} \quad (26)$$

where $\mathbf{P}^{\mathrm{T}}\Phi_{11}$ converges to $\mathbf{P}\phi(\Delta T)$.

6. Generate the error vector $\mathbf{E}^k$. Specifically, at the k-th iteration counter

$$\mathbf{E}^k = \begin{Bmatrix} \mathrm{x}(\Delta T) - \mathrm{P}\mathrm{x}(0) \\ \delta \end{Bmatrix}^k \quad (27)$$

where $\mathbf{x}(\Delta T)$ represents the solution vector at the end of ΔT and δ is the trim-error vector corresponding to the initial-condition vector s.

7. Improve the solution with Newton damping parameter χ:

$$\mathrm{s}^{k+1} = \mathrm{s}^k - \chi\Phi^{-1}\mathbf{E}^k \quad (28)$$

The instructions 3–7 are repeated till the convergence of control inputs and periodic responses.

Parallel Fast Shooting

The parallel fast-shooting algorithm is a self-scheduling MIMD algorithm and utilizes the widely used master-slave processor approach (*e.g.*, Ref. 10). A processor, designated as the master, partitions a task and/or data domain, distributes the various subtasks together with data to be operated upon to other active processors (designated as the slaves), and collects and assembles the results from the slaves. The slave processors receive and execute the assigned subtasks and return the results to the master. Moreover, the inherently sequential parts of the algorithm are performed by the master. Specifically, in the trim analysis, the master processor sets up the initial conditions for integration and distributes them among the slave processors, forms the Jacobian matrix using the solutions received from the slaves, and upgrades the trim values of control inputs and initial conditions for periodic responses at the end of each iteration. Similarly, each slave processor receives a set of initial conditions from the master processor, integrates the equations of motion, estimates the forces and moments, and sends these solutions to the master processor. Thus, in each iteration cycle, each column of the Jacobian is generated by one slave processor, and this process is repeated (if necessary) until all the columns of the Jacobian are generated. Figure 1 schematically represents these operations, which shows that the algorithm is composed of two parts. While the first part corresponds to the master processor, the second part to the slave processors; they are referred to as the master and slave part, respectively. As seen from Fig. 1, the master part contains the entire shooting algorithm, which is executed sequentially on the master processor until a parallelized step is reached. At that step, the execution control switches to the slave processors, which perform the computations and return the control to the master. Then, the master processor proceeds to the next step. For simplicity and generality, the calls for interprocessor communication are not shown explicitly in Fig. 1. In the present study, the algorithm is implemented using the MPI routines for interprocessor communication. Given this background, we present the instructions of the algorithm in the respective parts in a format that is implemented.

Master Part:

1. Assumes a vector s of size $N + c = M$, which represents a set of arbitrary initial conditions for structural and aerodynamic states as well as for control inputs.

2. Forms the permutation matrix $\mathbf{P}$.

3. Forms $(M + 1)$ sets of initial-condition vectors by perturbing only one element of s for each set of initial conditions. The $(M + 1)$-th set is a vector of unperturbed states; see Eq. (24).

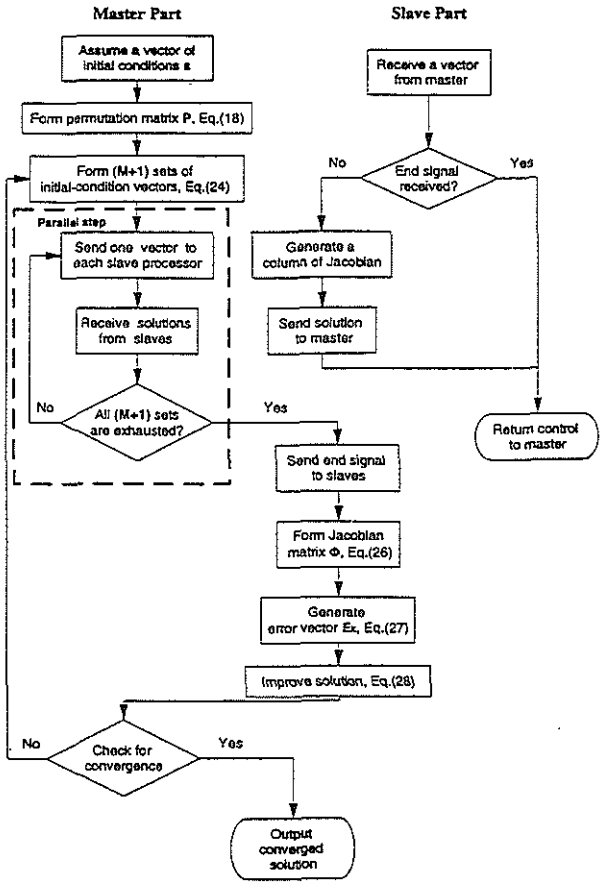

Figure 1: Schematic of Parallel Fast Shooting for MIMD Architecture

4. Sends as many sets of initial conditions as the available number of slave processors. For example, if $(M+1) > p_s$ (p_s = number of available slave processors), it sends the first p_s sets of initial-condition vectors.

5. Receives the solution vectors $\mathbf{y}^i$ $(i = 1, 2, \ldots M)$ or $\mathbf{y}$ (see Eq. (25)) at the end of the period ΔT from the slave processors and stores them.

6. Checks whether all sets of initial-condition vectors are processed. If not, identifies the slave processors that have completed their tasks and distributes the remaining sets of initial conditions till all the columns of the Jacobian are generated. Otherwise, sends an end signal to the slave processors.

7. Forms the Jacobian matrix Φ using Eq. (26).

8. Generates the error vector $\mathbf{E}_k$ from Eq. (27).

9. Improves the solution according to the Newton iteration (Eq. (28)).

Repeats steps 3–9 till convergence.

Slave Part:

1. Receives one set of initial conditions for the structural and aerodynamic states and control inputs from the master processor.

2. Integrates Eq. (7) through the time interval ΔT and computes response $\mathbf{x}(\Delta T)$ as well as trim-error δ. In other words, generates the solution vectors $\mathbf{y}^i$ $(i = 1, \cdots, M)$ or $\mathbf{y}$ (see Eq. (25)). Thus, each column of the Jacobian is generated by one processor.

3. Sends the solution vectors $\mathbf{y}^i$ or $\mathbf{y}$.

4. Repeats steps 1–3 till an end signal is received from the master processor and then exits to the master part.

Stability

Since the EFTM comes out as a byproduct of the trim analysis, the stability analysis involves determination of the modal damping levels and frequencies from the eigenvalues of the EFTM and involves hardly 1% of the total run time; see Eqs. (23a) and (23b). An LAPACK subroutine DGEEV for real unsymmetric matrices is used to compute the eigenvalues and eigenvectors of the EFTM (Ref. 14); in the master-slave processor approach, this routine is executed by the master processor; that is serially. As seen from Eq. (23b), the inverse arctangent function results in multiple values or nonunique frequencies. Therefore, to compute the frequencies and thereby identify the correct modes, the mode-identification method of Ref. 13 is used with the modification of replacing the eigenvalues and eigenvectors of the FTM with those of the EFTM. Specifically, for each eigenvector, say $\mathbf{x}_i$, the complex ratio of the derivative $\dot{x}_i$ and the state x_i corresponding to the most dominant component is computed. The imaginary part of this complex ratio, with a suitable correction, which is an integer multiple of Q (not an integer as in the conventional Floquet analysis; see also Ref. 13), closely approximates the frequency of the mode. Thus, the frequency indeterminacy is reduced by a factor of Q using the fast-Floquet analysis.

Computational Reliability

The trim and stability analyses require solutions for nonlinear differential equations of motion coupled with algebraic transcendental equations of trim. Solving such a complex system is computationally demanding and involves a very large number of numerical operations such as integrations of equations, linearizations for Newton improvement and repeated iterations with improved starting values. These computations are prone to numerical corruptions or small deviations

from the exact values due to round-off or discretization errors, which can get magnified and finally affect the trim and stability predictions. Therefore, it becomes necessary to quantify the computational reliability of the fast-Floquet analysis. Specifically, in the trim analysis, the Jacobian influences the convergence of trim results of control inputs and periodic responses; see Eqs. (26) and (28). Furthermore, as seen from Eq. (26), the EFTM is extracted from the Jacobian, and the stability results are generated from the eigenvalues and eigenvectors of the EFTM, which are also susceptible to numerical corruptions. Therefore, the computational reliability concerns both trim and stability analyses. Following earlier studies (*e.g.*, Ref. 8), we use three computational reliability parameters.

First, the condition number of the Jacobian matrix Φ, which is defined as

$$\text{Cond.}(\Phi) = \frac{\left[\text{max. eigenvalue of} \Phi^T\Phi\right]^{\frac{1}{2}}}{\left[\text{min. eigenvalue of} \Phi^T\Phi\right]^{\frac{1}{2}}} \tag{29}$$

Second, the condition number, Cond.(λ), for the eigenvalue of the mode of interest. Let $\mathbf{x}$ and $\mathbf{y}$, respectively, be the right and left eigenvectors of the EFTM corresponding to an eigenvalue λ; that is, $[\text{EFTM}]\mathbf{x} = \lambda\mathbf{x}$ and $[\text{EFTM}]^T\mathbf{y} = \lambda\mathbf{y}$. Then, Cond.($\lambda$) is computed from the expression

$$\text{Cond.}(\lambda) = |\mathbf{y}^T\mathbf{x}|^{-1} \tag{30}$$

Third, the residual error ε of the eigenpair $(\lambda, \mathbf{x})$ is given by

$$\varepsilon = \frac{\|[\text{EFTM}]\mathbf{x} - \lambda\mathbf{x}\|}{\|\lambda\mathbf{x}\|} \tag{31}$$

For additional details, see Ref. 4.

Performance Metrics

Measuring the performance of parallel algorithms is an important aspect of parallel computing. In general, the performance metrics provide a means of estimating the overall effectiveness of parallel algorithms on different computing systems, and for problems of varying size. Under ideal conditions, irrespective of the problem size, a perfect parallel algorithm implemented on a computing system with p processors is expected to reduce the run time by a factor of p. However, there are several limiting factors such as communication overhead and memory size, because of which the performance of a parallel algorithm deviates from the ideal. Following the literature (*e.g.*, Refs. 15 and 16), we use five performance metrics to measure the effectiveness of the parallel fast-Floquet analysis: run-time saving and its growth with the order, speedup, efficiency, serial and parallel fractions, and portability.

The run time and its variation with the order are of particular importance. They are directly measured quantities and the easiest to interpret. The parallel run time is defined as the total elapsed time from the beginning to the end of the execution of the parallel program. It includes computation time, communication time and processor idle time. The computation time depends mainly on the problem size and the number of processors and to a lesser extent, on the memory size and speed of the processors. The communication time is the time spent for message startup and transfer. It depends not only on the type, bandwidth and latency (time to initiate and complete a communication process) of the communication network, but also on the size and number of messages being communicated (Ref. 16). The processor idle time basically occurs due to the lack of computation or data contention during execution. To achieve a better performance, the processor idle time needs to be minimized; this can be done by properly balancing the workload among the processors and, wherever possible, by overlapping computation and communication. In this study, we use the MPI timer routine MPI_WTIME for measuring the parallel run time (Ref. 11). The other three metrics — speedup, efficiency, and serial and parallel fractions — are derived from the measured parallel run time and the predicted uniprocessor run time. The uniprocessor run time often is not a measured quantity owing to constraints in the architecture, algorithm or excessive run time. For example, the master-slave processor approach requires at least two processors and precludes a direct measurement of the uniprocessor run time. Under such circumstances, it has to be predicted.

Speedup S_p provides a measure of how a parallel algorithm executing on p processors reduces the run time of the same algorithm executing on one processor. If t_j and t_1 represent the run time on j processors and one processor, respectively, speedup is defined as

$$S_p = \frac{t_1}{t_p} \leq p \tag{32}$$

Similarly, efficiency E_p provides a measure of how well the processors are kept busy. In other words, it is a measure of effective utilization of the processors and is given by

$$E_p = \frac{S_p}{p} \leq 1 \tag{33}$$

Under ideal conditions, $S_p = p$ and $E_p = 1$. This means the algorithm is perfectly parallel and the 'best' the processors can do has been achieved. However, as seen from Eqs. (32) and (33), speedup and efficiency both depend upon the number of processors, and on the run time constraint with increasing model order. In practice, for a given model order, the saving in run time decreases with increasing number of processors. Therefore, while the speedup increases with increasing number of processors, efficiency drops progressively. Therefore, it is necessary to interpret the speedup and efficiency in a relative sense as a compromise between how fast a job needs to be completed (speedup) and how well the processors are utilized (efficiency).

In this study, we follow Ref. (17) to predict the uniprocessor run time. Basically, it is taken to be a sum of serial portion t_{1_s} and parallel portion t_{1_p}. The assumption is that the problem can be divided into a completely serial portion and a completely parallel portion, which can be divided equally among the processors. Thus, we have

$$t_1 = t_{1_s} + t_{1_p} \tag{34}$$

Since only the parallel portion can be speeded up, the expression for run time with p processors is

$$t_p = t_{1_s} + \frac{t_{1_p}}{p} \tag{35}$$

We measure a set of values for the parallel run time t_p by executing the same job on a different number of processors p. Then using Eq. (35), t_{1_s} and t_{1_p} are determined by following a least-square approach. Thus, the uniprocessor run time $t_1(= t_{1_s} + t_{1_p})$ and thereby speedup and efficiency are calculated; see also Eqs. (32) and (33).

Now, it is expedient to express t_{1_s} and t_{1_p} in terms of dimensionless serial fraction f. This is done in Eq. (36):

$$t_{1_s} = f t_1 \text{ and } t_{1_p} = (1-f)t_1 \tag{36}$$

Using Eq. (36) in Eq. (35), the parallel run time can be expressed as

$$t_p = f t_1 + \frac{(1-f)t_1}{p} \tag{37}$$

Therefore, the expression for speedup can be rewritten as

$$S_p = \frac{1}{f + \frac{1-f}{p}} \leq \frac{1}{f} \tag{38}$$

which is the well known Amdahl's law (Ref. 10). Basically, it demonstrates the significance of the serial fraction on the overall effectiveness of a parallel algorithm and shows that the speedup is limited by the reciprocal of f.

From Eq. (38), the serial fraction is expressed in terms of speedup and number of processors as

$$f = 1 - \frac{1 - 1/S_p}{1 - 1/p} \tag{39}$$

which shows f is strictly not independent of p. Although p takes only integer values, substituting Eq. (38) in Eq. (33) and differentiating $1/E_p$ with respect to p, we get

$$\frac{d}{dp}\left(\frac{1}{E_p}\right) = f \tag{40}$$

which shows that the serial fraction can be an indirect measure of efficiency.

Portability is a performance measure that cannot be quantified; nevertheless, it is very important in practical parallel computing. Portability is the ease with which the same parallel algorithm can be implemented on different machines/architectures. The parallel fast-Floquet algorithm developed in this study employs the MPI (Ref. 11). It is a standard message passing library for interprocessor communication, which facilitates development of portable algorithms as demonstrated by the implementation of the same parallel fast-Floquet analysis on both the massively parallel computer and distributed computing system of networked workstations.

Results

The results refer to the parallel performance metrics as well as computational reliability. Specifically, the performance metrics data include run time and its variation with the order and number of processors p, speedup, efficiency, rate of change of reciprocal of efficiency with respect to p, and serial and parallel fractions; see also Eqs. (32), (33), (36) and (40). The run time refers to the total elapsed time for generating both trim and stability results. Besides these metrics, we also address the portability of the parallel fast-Floquet algorithm on different computing systems. The computational reliability parameters comprise the condition number of the Jacobian in the converged cycle, condition number of the eigenvalue of the lag regressive-mode damping level and residual error of the corresponding eigenpair; they are computed according to Eqs. (29)–(31). The serial computations are performed on a mainframe VAX 4320 computer. The parallel computations are done on two types of hardware: a distributed computing system of 13 SUN SPARC networked workstations and a massively parallel IBM SP-2 computer. We reiterate that these systems follow MIMD architecture and, therefore, the same algorithm is implemented on both the systems using FORTRAN 77 and the MPI library for interprocessor communication (Ref. 11).

The results are generated for isolated-rotor models of hingeless helicopters in trimmed flight. The rotors have three and more blades undergoing rigid flap and lag motions and are isolated in that their support systems are rigid or stationary. The airfoil aerodynamics is based on the ONERA dynamic stall models of lift and drag, and the rotor downwash dynamics is represented by a finite-state three-dimensional wake model. Moreover, the trim conditions include the moment equilibrium (zero rolling and pitching moments) and the equilibrium of the longitudinal forces in the longitudinal-vertical plane of the rotor. For additional details on the equations of motion for structural and aerodynamic representations as well as for trim formulation, see Refs. 8 and 9. The blade is discretized with 10 aerodynamic elements with four dynamic stall states per element, and then the model order or number of states is controlled by varying the number of wake harmonics or wake states in modeling the wake.

Unless otherwise stated, the following baseline parameters are used: $\mu = 0.3$, $\gamma = 5$, $P_\beta = 1.15$, $\omega_\zeta = 1.14$, $\sigma = 0.05$, $a = 6.28$, $C_{d_0} = 0.0079$, $C_w = 0.00375$, and $\overline{f} = 0.01$. The results are generated for three models with $M = 227$, 329 and 395 and they are presented in two phases according to the type of hardware used. Specifically, Figs. 2–5 refer to a massively parallel IBM SP-2 computer, and Figs. 6–9 to a distributed computing system of networked workstations; this is followed by Fig. 10, which shows a summary of the run-time growth with the order on the serial and two types of parallel hardware.

To help appreciate the results, we recall that the standardization of the parallel performance metrics is still evolving, and distributed computing in particular is an emerging area of the past few years. Furthermore, the evaluation and interpretation of these metrics depend upon several factors such as the type of the problem and the parallel-computing hardware, and merit additional comments. According to Eqs. (32) and (33), the speedup S_p and efficiency E_p are derived metrics, and they depend on the measured parallel run time t_p as well as on the predicted uniprocessor run time t_1 for the parallel algorithm. The uniprocessor run time t_1 is predicted by following the assumption of Eq. (34) and using a series of measured values for t_p; this requires executing the same job with a fixed model order for a varying number of processors and then predicting t_1 by a least-square approach. This means t_1, S_p and E_p are strongly dependent on the problem and hardware; in particular, S_p and E_p are sensitive to the accuracy of predicting t_1. Therefore, it becomes necessary to exercise considerable care when comparing these performance metrics from the SP2 computer and the distributed computing system although the same algorithm is used on these two systems. Nevertheless, these metrics along with the measured values of run time and its variation with the order and number of processors collectively provide a means of assessing the overall effectiveness of the parallel fast-Floquet analysis on a specific hardware.

Figure 2 shows the variation of the run time with the number of processors for $2 \leq p \leq 64$. Overall, as expected, the run time decreases with increasing number of processors for a fixed model order and increases with increasing model order for a fixed number of processors. In particular, for the model with $M = 227$, the run time remains nearly constant for $p > 10$; that is, a further increase in the number of processors yields no appreciable saving in run time. For the larger models with $M = 329$ and 395, the rate of reduction in run time with p is significant only for $p \leq 16$, and the run time virtually flattens out for $p > 32$ or so. This lack of reduction in run time with increasing p is associated with delays in interprocessor communication. In IBM SP-2, communication takes place through a switch, which is much slower when compared to the

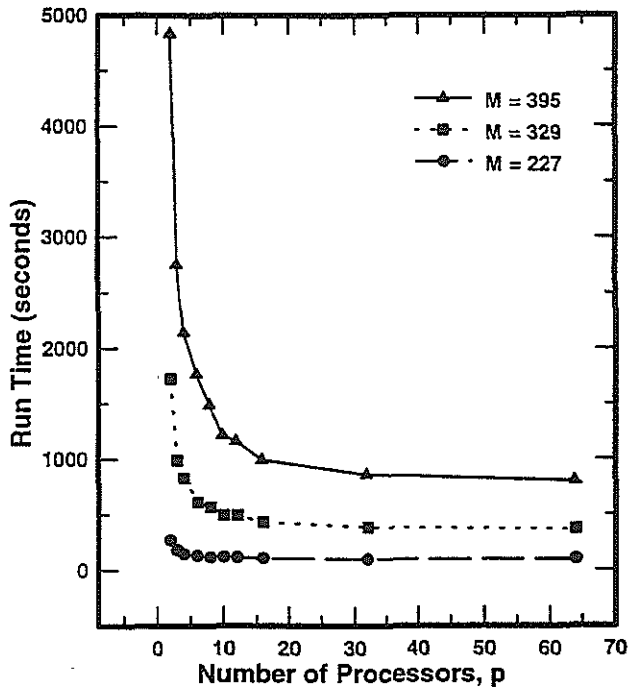

Figure 2: Run-Time Variations with the Order and Number of Processors on a MIMD Computer (IBM SP-2)

speed of individual processors. Thus, as the number of processors increases, the communication delays increase accordingly. This negates any computational gains achieved through parallelism. The impact of this communication overhead on speedup and efficiency is studied in Fig. 3, where the results are presented for $2 \leq p \leq 32$.

In Fig. 3, while part 'a' shows the variation of speedup with the number of processors and model order, part 'b' shows the corresponding variation of efficiency. As expected, for a fixed order with increasing number of processors, the speedup increases and efficiency decreases. However, for a fixed number of processors, both speedup and efficiency increase with increasing order M. This means if the job needs to be completed faster, it is necessary to increase the number of processors; see also Fig. 2. But this increase in speedup is accompanied by reduced efficiency. Figure 3 also shows that the speedup and efficiency figures are close to the ideal for $p \leq 6$ for $M = 329$ and 395. For example, the speedup and efficiency are, respectively, equal to 1.8 and 95% for $M = 395$ with $p = 2$, and with $p = 10$, a five-fold increase, the speedup increases to 7.5 and the efficiency comes down to 75%. However, in general, Fig. 3 shows that S_p and E_p deviate considerably from the ideal values with increasing number of processors ($p > 10$). This is related to idling of the processors and interprocessor communication. In the master-slave algorithm, the slave pro-

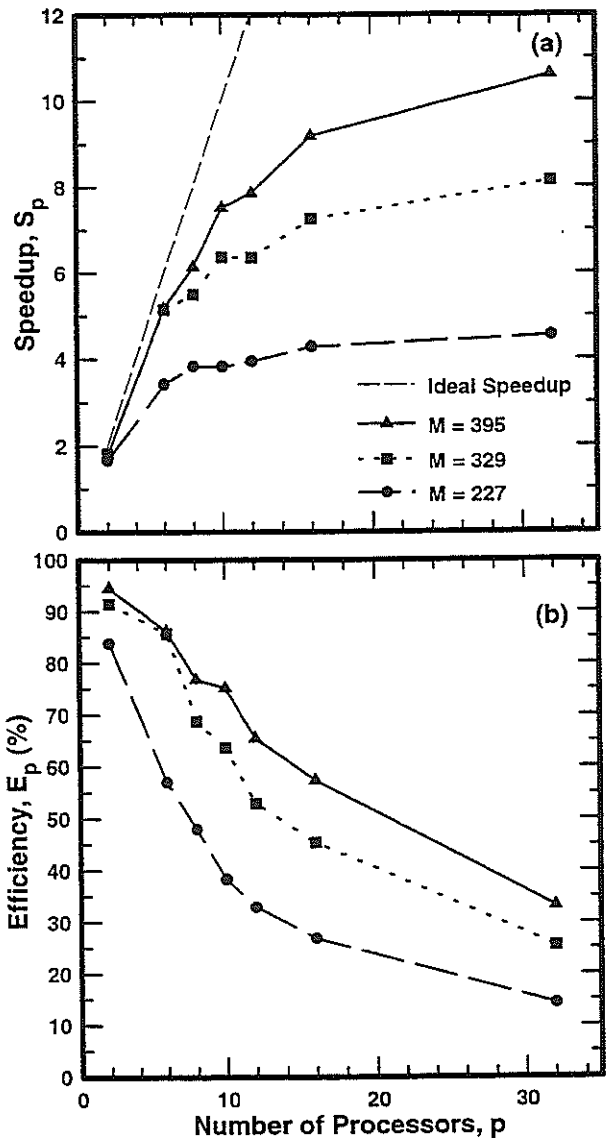

Figure 3: Speedup and Efficiency Variations with the Order and Number of Processors on a MIMD Computer (IBM SP-2)

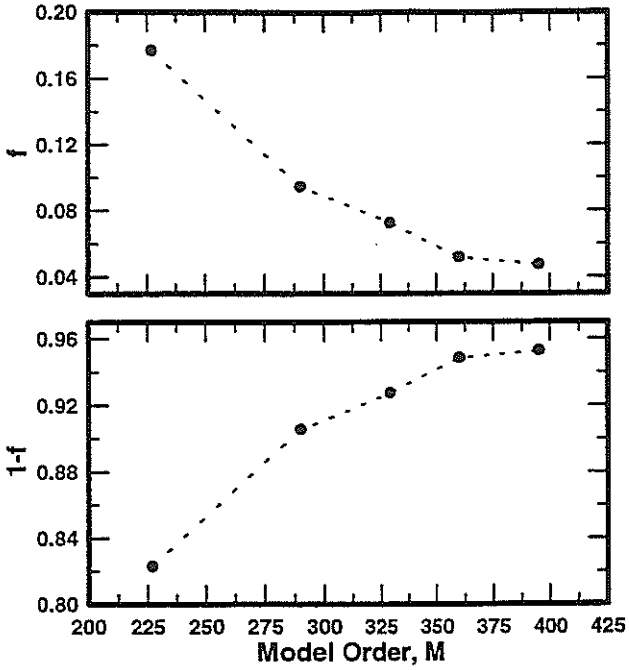

Figure 4: Variations of Serial and Parallel Fractions with the Order on a MIMD Computer (IBM SP-2)

cessors, which are larger in number, remain idle when the master executes the serial portion. Similarly, the master remains idle when the slaves are performing the parallel portion of the problem. Consequently, the estimated uniprocessor run time is dominated by the serial portion of the problem, which limits speedup; see Eqs. (36) and (38). Moreover, the computations involved in the simple rigid blade model is not large enough to fully exploit the enormous computing power of the SP-2 computer with $p > 10$.

The next two figures address the overall effectiveness of the parallel fast-Floquet analysis based on the serial and parallel fractions (Fig. 4) and on the rate of change of reciprocal of efficiency with respect to the number of processors (Fig. 5); see also Eqs. (36), (38) and (40).

Recall that the serial fraction limits the upper bound of speedup and can be an indirect measure of efficiency. As seen from Fig. 4, with increasing model order the serial fraction decreases or equivalently the parallel fraction increases. For example, the serial fraction, which is about 0.18 for $M = 227$ decreases to 0.045 for $M = 395$. Therefore, the upper bound of speedup increases from 5.5 to 20. This means the degree of parallelism increases with increasing M. This is expected as well; in the parallel fast-Floquet analysis, the bulk of the saving in run time is achieved by performing the repeated integrations in parallel, and the number of these integrations increases with increasing order. Similarly, Fig. 5 shows that the $1/E_p$-versus-p curve is approximately linear for all three models and that the slope of the curve decreases with increasing order. In other words, the serial fraction decreases and consequently efficiency increases with increasing M; see also Fig. 3.

In Figs. 6–9, we present the results from a distributed computing system. The computations are carried out on a network of 13 low-end SUN SPARC stations (IPC and LX). These workstations are heterogeneous in that the individual processors differ in memory and clock speed. Such architectural differences are dealt with efficiently by dynamically balancing the load among the processors through the master-slave processor algorithm. The workstations are accessible through a department-wide network; it is a local area network (LAN), which uses ethernet communica-

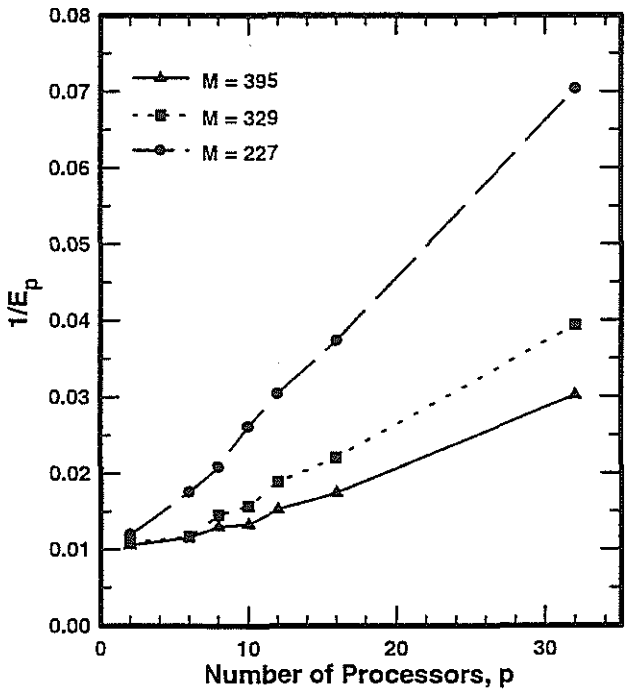

Figure 5: Variations of $1/E_p$ with the Order and Number of Processors on a MIMD Computer (IBM SP-2)

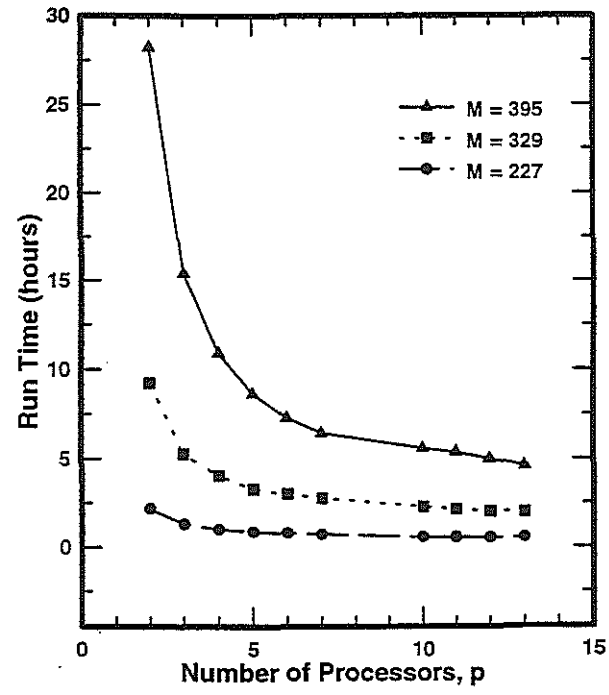

Figure 6: Run-Time Variations with the Order and Number of Processors on a Distributed Computing System

tion channel.

Figure 6 shows how the run time varies as we increase the number of processors and model order. For illustration, we consider the same three models treated earlier with $M = 227$, 329 and 395. As expected, the run time for a given model decreases with increasing number of processors and increases with increasing order. These features are similar to those observed on IBM SP-2 in Fig. 2, except that the run times are much longer. Moreover, the reduction in run time is appreciable, say for $p < 7$, for the model with $M = 395$ and to a lesser extent for the other two models. For $p > 7$, the run time for all three models either remains nearly constant or decreases only slightly. This means increasing the number of processors beyond a certain value does not yield a significant reduction in run time owing to communication overhead, which increases with increasing number of processors. In the distributed computing system of networked workstations, the interprocessor communication is through the ethernet communication channel, which has a small bandwidth (typically a few Mbits/sec) that remains the same even when more workstations are added to the network. By comparison, the processors of these workstations are sufficiently powerful in performing a few MFLOPS. Therefore, for large models with increasing number of processors, the communication channel can become a bottleneck with limited communication bandwidth and high latency. Moreover, in the master-slave algorithm, the master processor communicates with the slave processors $(M+1)$ times, and the length of the message to be communicated also increases with M. Furthermore, in a network of workstations, it is not always possible to perform the computations in a 100% dedicated fashion since the workstations have slight loads due to non-computational related operations, and the network is still being used by other workstations that are not actually participating in the computations. Thus, the overall performance of the distributed computing system is lower than what the system can nominally deliver. Nevertheless, we emphasize that the parallel code is completely portable from the massively parallel SP-2 computer to the distributed computing system and vice versa without any modification whatsoever.

The effectiveness of the parallel fast-Floquet analysis on a distributed computing system is further addressed in the next three figures, which, respectively, show the variation of the speedup and efficiency with increasing number of processors (Fig. 7), the serial and parallel fractions with increasing model order (Fig. 8), and $1/E_p$ curve with increasing number of processors (Fig. 9). As seen from Fig. 7, for a fixed model order, while the speedup increases with increasing number of processors, the efficiency basically decreases. Moreover, both speedup and efficiency increase with increasing order for a fixed number of processors. These results are similar to those obtained from IBM SP-2 computer (*e.g.*, Fig. 3). In particular, both speedup

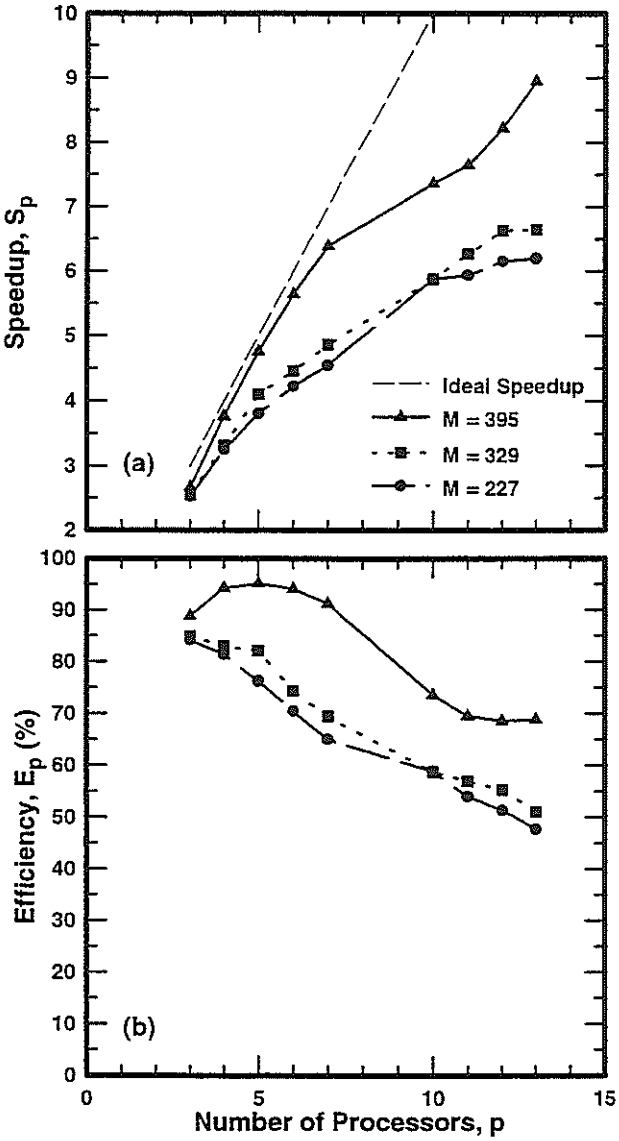

Figure 7: Speedup and Efficiency Variations with the Order and Number of Processors on a Distributed Computing System

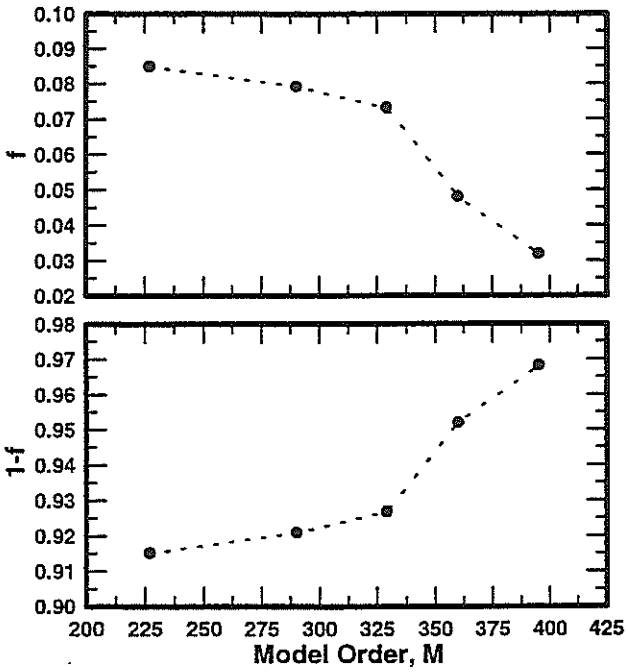

Figure 8: Variations of Serial and Parallel Fractions with the Order on a Distributed Computing System

and efficiency are close to the ideal values for $M = 395$ with $p \leq 7$, which is also the case for other two models for $p \leq 5$. Figure 8 basically shows that the parallel fraction is greater than 0.92 for the models considered; in fact, it is about 0.97 for $M = 395$ and consequently the serial fraction is a small number. This indicates that the fast-Floquet analysis is tailored to distributed computing as well. Figure 9 essentially corroborates the finding of Fig. 5 in that the slope of the $1/E_p$-versus-p curve decreases with increasing M, indicating that efficiency increases with M. However, it is seen that this curve has localized deviations from linearity, which merit further investigation.

In Fig. 10, we provide a comparison of run time and its growth with the order for the serial and parallel fast-Floquet analyses. Although the differences in the architecture and algorithm do not permit a direct quantitative comparison between the serial and parallel run times, a qualitative comparison is revealing in that it shows the necessity of turning to parallel computing. As seen therein, the serial run time grows between quadratically and cubically with the order ($\approx M^{2.4}$). Owing to this rapid growth, the run time is presented for relatively small order models ($94 \leq M \leq 169$). For example, the run time is 6 hours and 45 minutes for $M = 94$ and it increases to 2 days and 12 hours for $M = 169$. Such a run-time growth shows that the serial fast-Floquet analysis is not practical for routine treatment of models with hundreds of states despite the fact that it provides nearly a Q-fold reduction in run time of the conventional Floquet analysis (Ref. 7). However, Fig. 10 demonstrates the dramatic impact of parallelism on run time and its growth with the order, which varies from 79 to 395. Specifically, it shows that the run time for the fast-Floquet analysis is reduced dramatically. Moreover, the run times from both parallel implementations are much shorter and their rates of growth are much slower. To help appreciate this comparison, the scale of the ordinate is magnified in the inset of Fig. 10. It is seen that the run time from the implementation on IBM SP-2 grows very slowly. By comparison, the run time from the distributed computing system implementation is much longer and grows with a higher slope. For example, for $M = 329$, the parallel run times are 367 and 7145 seconds on the MIMD and distributed com-

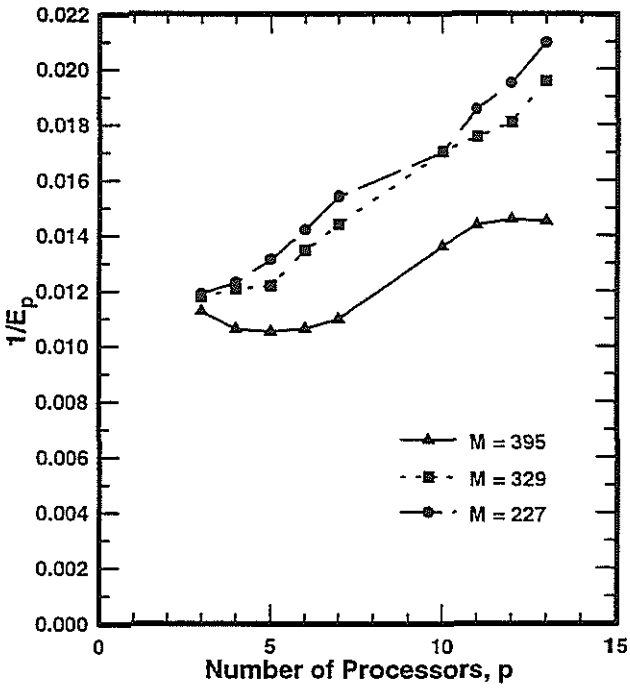

Figure 9: Variations of $1/E_p$ with the Order and Number of Processors on a Distributed Computing System

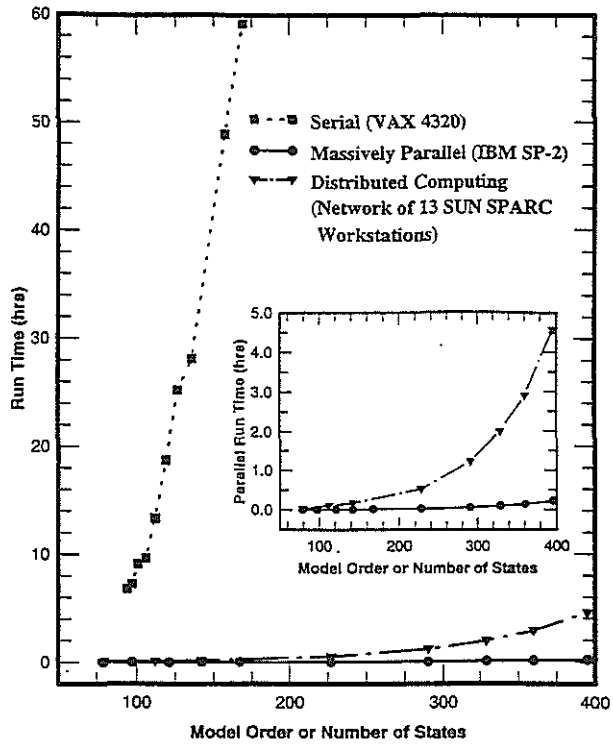

Figure 10: Run-Time Variations on Serial, Parallel and Distributed Computing Systems

puting systems, respectively. Such a comparison also shows that the distributed computing system can deliver a performance that is fairly comparable to that of a massively parallel computer for relatively large models, say $M < 300$.

Table 1 presents a sample of computational reliability parameters for the parallel fast-Floquet analysis from the IBM SP-2 computer and distributed computing system. It is seen that the reliability parameters from both the implementations are comparable. Moreover, the condition numbers of the Jacobian as well as the eigenvalue condition numbers are acceptable and the residual errors of the eigenpairs are negligible; for additional details, see Refs. 2 and 4.

Table 1: A Sample of Computational Reliability Parameters
M: Massively Parallel Computing
D: Distributed Computing

System Order, $N+c$	Condition number of the Jacobian matrix for the converged cycle	Eigenvalue condition number for the lag regressive mode	Residual error of the corresponding eigenpair
227 (M)	204351.57	2.5092	0.7254E-14
227 (D)	173326.28	2.5181	0.5247E-14
395 (M)	651771.24	2.6393	0.2812E-13
395 (D)	572671.30	2.6412	0.2223E-13

Conclusions

The preceding parallel fast-Floquet algorithm predicts trim and stability of rotors with identical and equally spaced blades. It exploits the fact that such a rotor with Q blades has Q planes of symmetry. It is designed for MIMD computing architecture, which is almost exclusively used in mainstream parallel-computing systems of networked workstations (distributed computing) and massively parallel computers. It is also portable in that it can be directly implemented on both systems. Models with hundreds of states are treated on these two systems as well as on a serial computer, and a comprehensive database is generated on run time and its growth with the order, turnaround time and computational reliability. The database also includes additional parallel-performance metrics such as speedup and efficiency, which, respectively, show how fast a job is completed with a set of processors and how effectively these processors are used. These data lead to the following findings:

1. The serial run time grows between quadratically and cubically with order. By comparison, both the parallel systems reduce the run time dramatically; in fact, the two ratios of serial run time versus the two parallel run times rapidly increase with increasing order. More importantly, both the parallel implementations provide a means of controlling the growth of run time with the order by

a judicious combination of speedup and efficiency; that is, increasing the number of processors with the order.

2. The parallel-performance data of speedup, efficiency and parallel fractions from the two parallel implementations are comparable; so are the computational reliability figures from the serial and two parallel implementations. In particular, the speedup and efficiency figures are close to the ideal values for some combinations of model order and number of processors; as expected, the run time on networked workstations is much longer than that on a massively parallel computer for very large models, say for $M > 300$.

3. With respect to developing a parallel algorithm and turnaround time, treating a large model with hundreds of states on networked workstations is as routine as treating a small model ($N < 100$) on a workstation. This is a measure of the practical utility of distributed computing in treating large models and offers considerable promise for comprehensive- and design-analysis applications.

Acknowledgment

This work is sponsored by the Army Research Office (Grant DAAH04-94-G0185). NASA Ames Research Center, Moffett Field, CA, Maui High Performance Computing Center, Maui, HI, and Cornell Theory Center, Ithaca, NY, sponsored computer time on their IBM SP-2 systems.

References

1. Gaonkar, G. H. and Peters, D. A., "Review of Floquet Theory in Stability and Response Analysis of Dynamic Systems with Periodic Coefficients," The R. L. Bisplinghoff Memorial Symposium Volume on Recent Trends in Aeroelasticity, Structures and Structural Dynamics, University Presses of Florida, 1987, pp. 101–119.

2. Achar, N. S. and Gaonkar, G. H., "Helicopter Trim Analysis by Shooting and Finite Element Methods with Optimally Damped Newton Iterations," *AIAA Journal*, Vol. 31, 1993, pp. 225-234.

3. Achar, N. S. and Gaonkar, G. H., "An Exploratory Study of a Subspace Iteration Method as an Alternative to the QR Method for Floquet Eigenanalysis," *Journal of Mathematical and Computer Modeling*, Vol. 19, 1994, pp. 69-73.

4. Ravichandran, S., Gaonkar, G. H., Nagabhushanam, J. and Reddy, T. S. R., "A Study of Symbolic Processing and Computational Aspects in Helicopter Dynamics," *Journal of Sound and Vibration*, Vol. 137, 1990, pp. 495-507.

5. Peters, D. A., "Fast Floquet Theory and Trim for Multi-Bladed Rotorcraft," Proceedings of the 51st Annual Forum of the American Helicopter Society, Fort Worth, TX, 1995, pp. 444-459.

6. McVicar, J. S. G. and Bradley, R., "Robust and Efficient Trimming Algorithm for Application to Advanced Mathematical Models of Rotorcraft," *Journal of Aircraft*, Vol. 32, 1995, pp. 439-442.

7. Chunduru, S. J., "*Dynamic Stall and Three-Dimensional Wake Effects on Trim, Stability and Loads of Hingeless Rotors with Fast Floquet Theory*, Ph.D. Thesis, College of Engineering, Florida Atlantic University, Boca Raton, FL, 1995.

8. Subramanian, S., Gaonkar, G. H., Nakadai, R. M. and Nagabhushanam, J., "Parallel Computing Concepts and Methods for Floquet Analysis of Helicopter Trim and Stability," *Journal of the American Helicopter Society*, Vol. 41, 1996, pp. 370-382.

9. Subramanian, S. and Gaonkar, G. H., "Parallel Fast-Floquet Analysis of Trim and Stability for Large Helicopter Models," Proceedings of the 22*nd* European Rotorcraft Forum and 13*th* European Helicopter Association Symposium, Brighton, UK, September 1996, pp. 94.1-94.14.

10. Kumar, V., Grama, A., Gupta, A. and Karypis, G., *Introduction to Parallel Computing, Design and Analysis of Algorithms*, Benjamin/Cummings Publishing Company, New York, 1994.

11. Snir, M., Otto, S., Huss-Lederman, S., Walker, D. and Dongarra, J. J., *MPI: The Complete Reference*, The MIT Press, Cambridge, MA, 1996.

12. Pfister, G. F., *In Search of Clusters — The Coming Battle in Lowly Parallel Computing*, Prentice Hall, New Jersey, 1995.

13. Nagabhushanam, J. and Gaonkar, G. H., "Automatic Identification of Modal Damping from Floquet Analysis," *Journal of the American Helicopter Society*, Vol. 40, 1995, pp. 39-42.

14. Anderson, E. et al., *LAPACK Users's Guide*, Society for Industrial and Applied Mathematics Publication, 1992, Chapters 2 and 3.

15. Karp, A. H. and Flatt, H. P., "Measuring Parallel Processor Performance," *Communications of the Association for Computing Machinery*, Vol. 33, 1990, pp. 539-543.

16. Foster, I., *Designing and Building Parallel Programs — Concepts and Tools for Parallel Software Engineering*, Addison-Wesley Publishing Company, Massachusetts, 1995.

17. Morse, S. H., *Practical Parallel Computing*, Academic Press, New York, 1994.