University of Glasgow

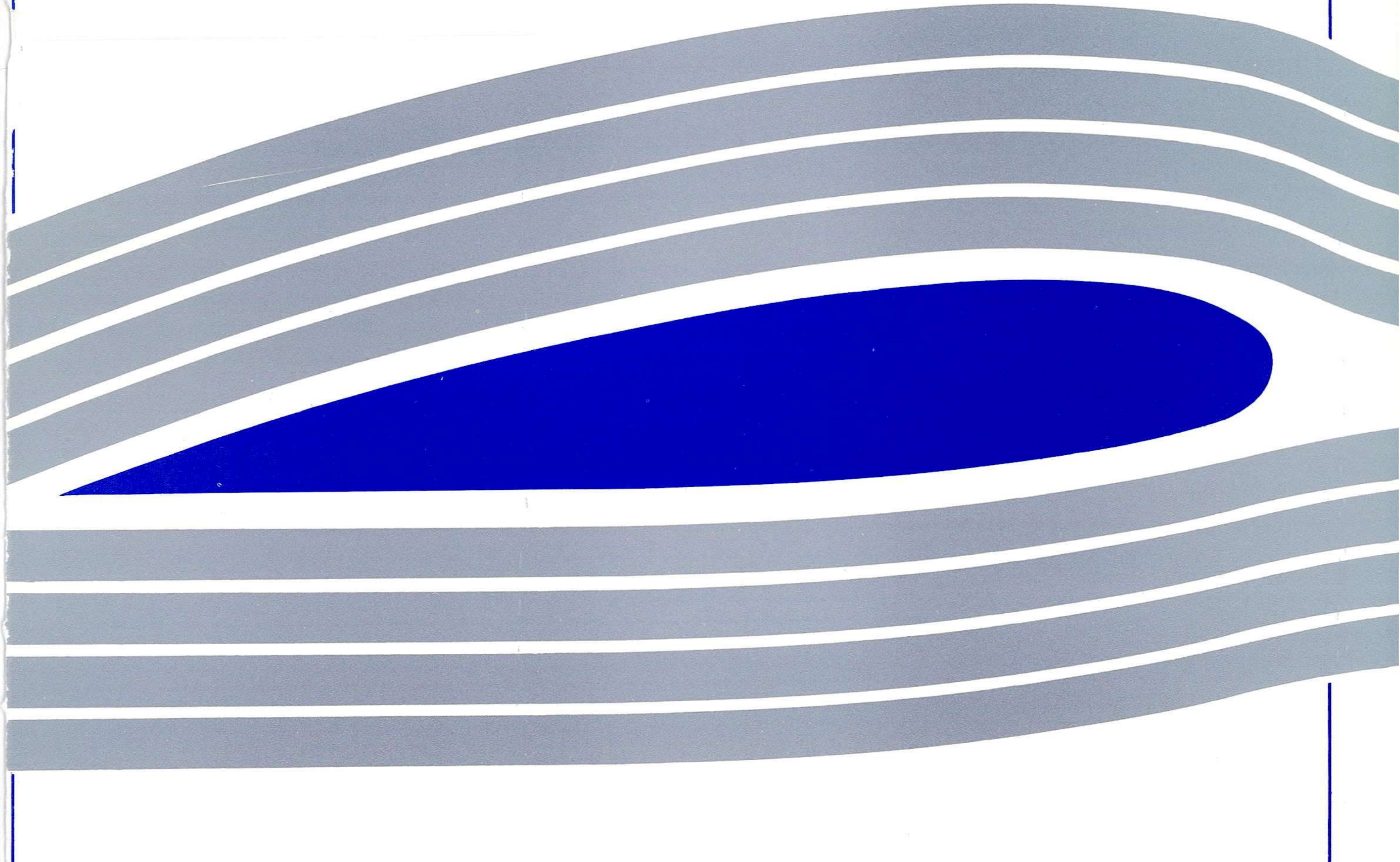
DEPARTMENT OF

AEROSPACE ENGINEERING

The Viability Of Parallel Processing Technology For Use In Rotorcraft Simulation

David H. Ewing

Internal Report No. 9504
March 1995



The Viability Of Parallel Processing Technology For Use In Rotorcraft Simulation

David H. Ewing

Internal Report No. 9504
March 1995

Summary

This report documents the viability of parallel processing technology for use in Rotorcraft Simulation. The generic rotorcraft simulation model RASCAL (Rotorcraft Aeromechanics Simulation for Control Analysis) has been 'parallelised' in a number of different ways and the computational performance has been measured and compared to that of the sequential code. The model has been implemented on the Parsytec series of parallel computers running PARIX and also on a cluster of Silicon Graphics Indy Workstations running PVM. The clustered workstation facility forms the HNW project funded by JISC NTI.

Contents

Nomenclature

A	matrix of co-efficients
I	identity matrix
T	transformation matrix
p	roll rate
q	pitch rate
r	yaw rate
$\underline{a}$	acceleration vector
$\underline{r}$	position vector
$\underline{u}$	control vector/velocity vector (Appendix 2)
$\underline{x}$	state vector
β	blade flap angle
ζ	blade lag angle
θ	blade pitch angle
ϕ_s	lateral shaft tilt
θ_s	longitudinal shaft tilt
$\underline{\omega}$	angular velocity vector

1. Introduction

1.1 The Requirement

It has been indicated that 25-75% of development flight test time is required to address flight dynamics deficiencies in the design (1). In order for these costs to be minimised, flight dynamics models could be used at an influentially early point in the design thus designing out areas of potential shortfall before a large financial commitment has been made. In order for such models to be used with confidence they must be able to appropriately portray the behaviour of rotorcraft that they represent.

One advancement in model fidelity that is of great consequence is the modelling of the most influential component in any rotorcraft - the rotor system. Until recently the rotor has been modelled by representing the rotor as an infinite number of blades i.e. as a disc. More recently, the rotor has been modelled more exactly - that is, as a finite number of blades each of which display degrees of freedom in flap, lag and feather. This allows periodic forces and moments to be evaluated as the rotor rotates about the azimuth. This advancement in modelling certainly achieves greater fidelity but pays the corresponding computational price.

In order to incorporate high fidelity rotorcraft simulation at an early stage in the design process the models must exhibit economical computational performance. One way in which it is possible to enhance the computational performance of the simulation is to use parallel processing. This involves using multiple computational nodes which execute different elements of the model in parallel thus reducing the overall run-time.

This report describes the parallel implementation and performance of one such rotorcraft simulation.

1.2 The Hardware

Parallel processing can be performed in two distinct ways: by multi-processor parallel computers and by clustered workstations. In the multi-processor machines processors are internally 'hard-linked' to allow message passing between nodes whereas clustered single processor machines communicate using Ethernet cable.

The multi-processor machines used in this investigation are the Parsytec series of parallel computers - a Supercluster, a Multicluster and an X'Plorer. The Supercluster machine has 64 T800 transputers, the Multicluster has 32 T800 transputers and the X'Plorer has 4 PowerPC processors.

The workstation cluster consists of 8 Silicon Graphics Indy workstations with R4400 processors. This cluster is part of a pilot project in parallel cluster computing and it is expected that the cluster will grow in size and also include machines from a range of vendors.

1.3 The Software

The Parsytec machines use the native message passing software entitled PARIX (PARallel extensions to UnIX). This software provides additional Unix commands to allow the compilation, execution, debugging and optimisation of parallel programs. It also provides high level language extensions to enable processors to uniquely identify themselves and to communicate with each other. The number of processors used in the execution of PARIX programs is determined by the user at run-time and a full copy of the code is loaded on to each processor. PARIX functions can then be used to define each processors identification and then Virtual Links are created between processors which will be required to communicate. Code can be executed in parallel using the processors identification and communication between nodes can take place. PARIX users have dedicated use of processors during the execution of the programs - no multi-tasking takes place.

The Silicon Graphics cluster uses different message passing software entitled PVM (Parallel Virtual Machine). This software provides commands and message passing functions similar to that of PARIX but the code executes in a different way altogether. A network of all the desired machines is created by the user and a master program is executed on one of the machines, typically this master task 'spawns off' sub-programs which are distributed amongst the machines in the network. These tasks will be distributed in accordance to whatever load balancing software is in operation. The sub-programs will each execute as a unique Unix task and each task can communicate with any other. PVM programs run within the multi-user, multi-tasking Unix environment so users do not benefit from dedicated processor use.

References 2 and 3 detail the workings of PARIX and PVM respectively and a description of some of the most common FORTRAN message passing functions is given in Appendix 1.

1.4 RASCAL Model

The simulation model used to investigate the impact of parallel processing is entitled RASCAL (Rotorcraft Aeromechanics Simulation for Control Analysis). A full mathematical description of RASCAL is given in ref 4. The modelling principles can be summarised as follows.

The model works by numerical time integration of the full state vector of the vehicle. The time derivative of each state is calculated via the translational and rotational equations of motion, airframe kinematics, blade flap, lag and feather equations of motion, and engine and wake dynamics. This forms the heart of the simulation.

The model can be trimmed to a prescribed flight state by minimisation of the vehicles accelerations using a function minimisation algorithm, the aerodynamic derivatives can be evaluated using small perturbation techniques and the free response to control perturbations or gust can be calculated.

The key features of RASCAL is that both rotors are modelled as a finite number of individual blades rather than as discs. This allows greater fidelity as complex blade shapes can be modelled using blade element theory and it also allows blade degrees of freedom to be incorporated. The blade kinematics are modelled precisely using current time-step states and not those of the previous time-step as is often incorporated in rotorcraft simulation. This allows the blade flap, lag and feather characteristics to be evaluated using the appropriate information.

1.5 Constraints

In order for the simulation to be used and further developed by non-specialist 'parallel programmers' the following constraints were required.

Where possible, the input and output of the code should be similar to that of the original sequential code and also that there is flexibility in the number of processors used (this is only applicable to the Parsytec computers). This should ensure 'user-transparency' to the parallel elements.

Where possible, the actual code should be similar to the original sequential code. This can be achieved by incorporation of a parallel layer which lies on top of the original code. This method also facilitates the speedy conversion of the code from one message passing language to another for example, the PARIX layer can be peeled off and replaced by the PVM layer. As new message passing libraries become available they will also be able to be easily incorporated. This should ensure 'developer-transparency' to the parallel elements.

2. Parallel Implementation

2.1 Trim

The starting point of any flight mechanics analysis is to obtain knowledge of the trimmed control states for a prescribed flight condition.

In RASCAL, this condition is obtained by minimising the vehicles accelerations to produce a quasi-trim state. A full description of the trimming method is described in ref 5. The trimmed state does not describe the periodic flight path that would be present - it works by setting the mean accelerations to zero. This is done by forward numerical integration to allow the transient forces and moments to decay and then the mean accelerations are calculated over the period. The model uses a NAG routine (E04FCF) to do the Newton-Raphson iteration required.

The algorithm can be parallelised in two distinct ways.

Firstly each time-step can be parallelised. That is, a number of processors could be used to split up the computation that was required during each integration step. The forces and moments from each blade and from each airframe component are calculated separately and then super-imposed to evaluate the overall forces and moments that exist on the vehicle. Scope for parallelisation exists by computing each of the contributory forces and moments separately (in parallel). With knowledge of the forces and moments present the state rates can be found via the translational, angular, flap , lag and feather equations of motion. The integration can then be performed and the process repeated.

Secondly, it would be possible to parallelise the trimming algorithm itself. Newton-Raphson techniques in rotorcraft simulation operate by evaluating an output vector, in this case the mean accelerations for a given set of controls and if the output vector is outwith a given tolerance each control is independently perturbed positively and negatively and a Jacobian matrix is constructed which is then used to provide the next estimate of the required controls. Each perturbed control is evaluated independently so scope for parallelisation exists here. This method involves removing the NAG routine and replacing it with the GENISA algorithm, a full description of which can be found in ref 6.

2.1.1 Time-step Parallelisation

2.1.1.1 Description

RASCAL is a sophisticated simulation which models both rotors as a finite number of individual blades, it also uses current time-step accelerations in the evaluation of the blades' inertial forces and moments. This means that the state equation formulation has added complexity. We can say that:

$$\underline{a}_{blade} = f(\underline{x}, \underline{u}, \dot{\underline{x}}, \dot{\underline{u}})$$

The total state vector thus has contributions that are independent of itself and contributions that are dependent. It is convenient then to describe the state vector in the following form:

$$\underline{x}_{total} = \underline{x}_{independent} + A' \underline{x}_{dependent}$$

Thus

$$\underline{x}_{total} = A^{-1}\underline{x}_{independent} \quad (1)$$

Where

$$A = I - A'$$

The computation which is required during each time-step is as follows.

Rotor aerodynamic calculations - the local velocity of a blade element is required which is used to evaluate the blades' aerodynamic forces and moments. The local velocity is a function of the aircraft's' velocity, rotorspeed, rates of flap, lag and feather and the velocity due to the wake. The wake model used in RASCAL is based on momentum theory.

Rotor inertial forces - the local acceleration of a blade element is required which is used to evaluate the blades' inertial forces and moments. The local acceleration is a function of the aircraft's' acceleration and angular velocity, rate of change of rotorspeed and the angular accelerations of the blades.

Engine calculations - the engine dynamics are function of the total rotor yawing moments.

Airframe dynamics - the aerodynamic forces and moments from the tailplane, fins and fuselage are evaluated from the local velocity.

These forces an moments are then used to solve the equations of motion, via inversion of the matrix A. It should be noted that is matrix is sparse and almost singular and prone to rounding errors. A routine to solve equation (1) directly has yet to be found and inversion via NAG routine F01AAF has proved the most robust to date. The state can then be integrated numerically and the process repeated until the full period is described.

The time-step parallelisation could be implemented in the following manner. One processor per blade could be used to evaluate the forces and moments on that blade and also the co-efficients of the A matrix which are pertinent to that blade. One other processor could be used to evaluate the airframe components forces and moments - as evaluation of the individual airframe components contributions can be executed in a short time relative to the blade calculations only one processor is used to evaluate the contribution from the fin, tailplane and fuselage. As the matrix inversion is a NAG routine and no other suitable means of solving the set of equations this part is to be left as sequential code. The numerical integration can be split between the processors with each 'blade' processor doing its own states integration. The other processor does the airframe states. This implementation is described pictorially in figure 1.

The message passing necessary in this implementation is plentiful. Before the simulation can begin each process requires knowledge of the flight condition and the helicopter configuration. So one processor reads the relevant information and broadcasts it to all the others.

During the execution of the code the following message passing is required at each time step. Consider first the 'blade' processors. The wake model used in RASCAL is a disc-type approximation and thus the blades' z-force and pitching and rolling moments need to be summed for each rotor system i.e. each processor will originally only have knowledge of say, the rolling moment of that blade but in order for the wake model to be evaluated that processor needs information about the rolling moments of the other blades in its rotor system. The engine dynamics also require knowledge of the yawing moments of the rotor system so this information has to be message passed and summed also.

During the calculation of the blades forces and moments the remaining processor calculates the airframe component dynamics and then receives the blade information which includes the co-efficients of the A-matrix. This processor then computes the equations of motion and inverts the A-matrix and solves for the total rates of states.

The blade pitch, flap and roll accelerations and rates are then sent to the appropriate blade processors and each blade processor performs the numerical integration of its own states. The aircraft states i.e. the translational and angular velocity, position and attitude and also the engine and wake states are computed on the remaining processor and the appropriate message passing is done which will allow progression to the next time-step with knowledge of the appropriate new states.

2.1.1.2 Results

The relative parallel performance improvement is described in table 1. The times are measured by accessing the internal clocks, **timenowlow** in the PARIX application and **etime** in the PVM application. It should be noted that etime returns the elapsed CPU time so the table assumes that the network contains at least the same number of machines as there are spawned tasks. This is assumed in all of the other implementations as well as this gives the clearest insight into the parallelism of each implementation. In order for this performance to be achieved there must be only one spawned task per node and each spawned task is receiving dedicated CPU usage. This is not necessarily the case but it is readily achievable through careful system administration if so desired.

In order to measure the performance improvement it was first necessary to evaluate the time to trim on a single processor. A typical case was chosen which trims the PUMA helicopter at 80 knots.

	T800	R4400
1 Node	240660s	2524s
10 Nodes	105552s	2726s
Speed-up	2.28	0.926

Table 1

In this application, the R4400 node is approximately 100 times faster than the T800 but it is the speed-up that is of concern as this best describes the relative parallel performance. It should also be noted that for consistency all of the T800 results are for Supercluster nodes. These have poorer performance than the Multicluster nodes and much poorer performance than the PowerPC nodes present in the X'Plorer.

2.1.1.3 Discussion

The parallel performance of this implementation displays disappointing results. There is only a small improvement in the run-time performance of the code on a parallel computer and when implemented on clustered workstations the code actually takes longer to execute in parallel than it does sequentially. The reasons for this are:

1. This implementation requires a large amount of message passing during the execution of each time-step of the code. This message passing suffers a significant overhead due to the extra processing required and also the associated idle time as processes wait for messages before computation can restart. The large amount of message passing required also explains the parallel performance difference between the parallel computer and clustered

workstations. The difference is two-fold: the parallel computer is a 'tightly-coupled' machine which is design specifically to have high communication performance, clustered workstations on the other hand use Ethernet cable to pass messages between one node and another; and the parallel performance improvement for implementations with a high degree of message passing is very dependent on the ratio of communications to computations. The Parsytec machines have relatively slow computational speeds but high communication speeds so are more suited to 'fine-grained' implementations - where only relatively small amounts of computation is performed with respect to the amount of message passing required. The Silicon Graphics nodes on the other hand, display excellent computational speeds but relatively slow communication performance so this type of platform is not so well suited to this sort of implementation.

2. The A-matrix inversion is computed on one processor and the other processors lie idle during this time. By profiling the code it was shown that this inversion takes around 32% of the CPU time in the sequential code. This situation could be improved in a number of different ways: direct solution of the state rate vector could be utilised if a robust method was implemented, a parallel solution to the inversion could be implemented or the inertial accelerations could be remodelled using information from the previous time-step thus eliminating the necessity for the existence of the matrix. This remodelling of the accelerations is described in section 2.3.1.

3. The 'blade' processors perform code which is merely rotor dependent for example the velocities and accelerations of the hub are calculated in duplicity. In the sequential code these were calculated only once. It is more efficient to calculate them in duplicity as this does not carry a message passing overhead. The other processors would merely be idle during this calculation anyway.

4. The wake model used is inconsistent with the rest of the code as it is a disc type model and not an individual blade model. The additional message passing that is required is the rotor-summed values of aerodynamic z-force and pitching and rolling moments.

5. Each blade task take longer than the airframe aerodynamic contributions so idle time occurs on that processor as it waits for information.

It is possible to place the wake model on the processor which computes the airframe aerodynamics but this displays no significant performance improvement. Similarly, it is possible to place the engine model on the 'blade' processors but this again results in no significant performance improvement.

There exists scope for additional parallelisation by separating the aerodynamic and inertial rotor forces and moments. The blade elements could also be split on to more processors. Neither of these would be able to provide significant performance improvement as only small amounts of code could be executed in parallel and much additional message passing would be incurred. All this would succeed in doing would be to make the implementation more finely-grained. None of the above reasons for the poor performance improvement would be addressed.

The PARIX implementation of the code will only run on the correct number of nodes or above. This sacrifices the user-transparency somewhat but was required in order to maintain the generic properties of the RASCAL model.

2.1.2 Trim Algorithm Parallelisation

2.1.2.1 Description

The NAG algorithm E04FCF was replaced by the inverse simulation algorithm entitled GENISA which is documented in ref 6. The algorithm solves for a set of initial conditions that produce a specified output. In the case of solving the trim condition for the RASCAL model the initial conditions are the control states and the output is the mean accelerations over the period, which, for the trim case, are zero.

The algorithm operates in the following manner. The initial (estimated) control state is used in the forward simulation to evaluate the output vector and if this output is outwith a predefined tolerance then each of the control states is perturbed positively and negatively in turn and the Jacobian matrix can be constructed using the rate of change of the output vector with respect to the control perturbation. This is then used to obtain an updated estimate of the controls which are then used in the forward simulation and the output vector is once again checked against the tolerance. If necessary the process is repeated until the convergence criteria are met.

Each of the perturbed states forward simulations are entirely independent of each other so they can be performed in parallel. The parallel execution of the code is described in figure 2.

2.1.2.2 Results

The performance of the trim algorithm parallelisation is described in table 2.

	T800	R4400
1 node	180863s	1947s
13 nodes	26212s	290s
Speed-up	6.90	6.71

Table 2

2.1.2.3 Discussion

This parallel implementation shows a much better performance improvement. The performance improvement on clustered workstations is comparable to that on the parallel computer, this is because of the limited message passing involved. The implementation is a 'coarser-grained' implementation than that of the time-step implementation so it does not suffer the high message passing overhead that was present. Also each of the perturbed states forward simulation perform exactly the same code - albeit with slightly different initial conditions, so the time for execution of the code is the same for each.

The reason that non-perfect speed-up is obtained is that the 'tolerance checking' forward simulation is executed on its own. That is, there is no scope to run any other code whilst the check is done. On the parallel computer where there is single-user access to the nodes the CPU cycles that are unused are wasted but on the multi-user, multi-tasking clustered workstations the CPU cycles are not necessarily wasted as they can be utilised by other users.

Each row of the Jacobian matrix is calculated by using double-sided numerical differentiation so the number of forward simulations required is two times the number of control states. Another forward simulation, using the unperturbed state, is required to evaluate the error vector. In the case of the PUMA helicopter and the trim state being calculated in a similar manner as before, there are 13 independent forward simulations. The algorithm then calculates a new control state and it runs the forward simulation to check if this new state is within tolerance. This forward simulation is done whilst the remaining processes are idle. In this case, there are a total of 14 forward simulations and the parallel implementation one process is to do is 2 forward simulations, the others only do one. So it is to be expected that the speed-up is of the order of 7.

The achievable speed-up is thus of the order of the number of control states plus one. The parallelisation will be directly applicable to the inverse simulation for which the algorithm was originally constructed. In inverse simulation the number of control states used for the PUMA is 4 so a speed-up of 5 can be expected.

The real parallel performance improvement can be obtained in a system which perturbs many states. One such system would be a periodic trimmer which perturbed the full state vector. One such trimmer is described in ref 7. It should be noted that the model used in ref 7 is implemented in parallel using a time-step implementation - the trimming algorithm itself is not parallelised. In the case of the PUMA which is described by 78 states there is a vast potential parallel performance improvement.

2.2 Small perturbation model

The small perturbation model is the classical way it examine an aircraft's control and stability characteristics. It is currently implemented as a single-sided differencing method but could be extended to a double-sided one. The aircraft's flight and control states are perturbed and the resulting accelerations are calculated.

The method can be parallelised in two ways: by parallelising the time-step and by computing each of the perturbed states in parallel.

2.2.1 Time-step parallelisation

2.2.1.1 Description

This method is similar to the one described in section 2.1.1.1

2.2.1.2 Results

The parallel performance improvement is shown in table 3.

	T800	R4400
1 Node	82611s	852s
10 Nodes	31945s	751s
Speed-up	2.59	1.13

Table 3

2.2.1.3 Discussion

The performance improvement from time-step parallelisation is similar to that described in section 2.1.1.3.

2.2.2. Perturbation Parallelisation

2.2.2.1 Description

Each perturbation is entirely independent of any other so the code can be parallelised in a very straightforward manner. Processors can each calculate the resultant accelerations from one perturbation. This parallelisation is described pictorially in Figure 3.

For the PARIX implementation it is also possible to implement the code in a manner that it will run successfully on any number of processors. This is in keeping with the user-transparency constraint.

2.2.2.2 Results

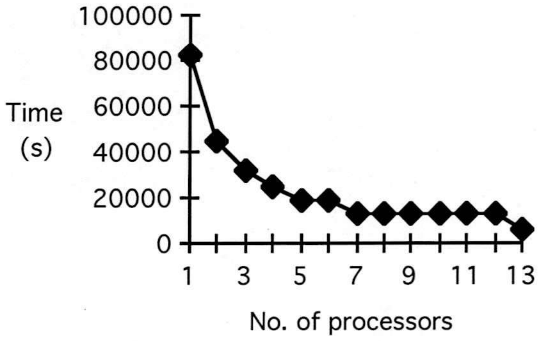
The parallel performance improvement is described in table 4.

	T800	R440
1 Node	82611s	852s
13 Nodes	6301s	68s
Speed-up	13.11	12.53

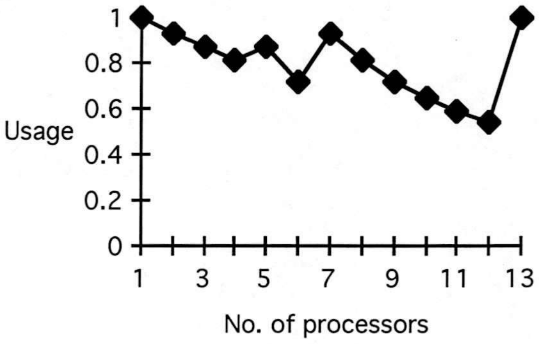
Table 4

It should be noted that the measured run-times are susceptible to congestion within the system. For example, the time taken to write to a file is dependent on other users. This explains the apparent anomaly of the T800 speed-up shown above.

The PARIX implementation can also be run on fewer than thirteen nodes. This is achieved by getting the nodes to do more than one perturbation. The performance improvement over varying numbers of nodes is described in Graph 1 and the average usage of the varying number of nodes is described in Graph 2.



Graph 1



Graph 2

2.2.2.3 Discussion

The perturbation parallelisation provides an excellent parallel performance improvement.

The parallel computer and clustered workstations show comparable parallel performance.

The code runs with near perfect speed-up on the correct number of nodes. In the PARIX application with fewer nodes than required the code also shows excellent performance improvement but it is susceptible to 'mis-matching' as the number of tasks often does not fit the number of nodes.

This implementation is a 'coarser-grained' one the that of the time-step implementation so it has similar benefits to that described in section 2.1.2.3.

The available parallelism is limited by the number of perturbed states and by the method deployed. For the PUMA case, using single-sided differentiation, the maximum number of nodes that can be usefully used is 13. However, 26 nodes could be used if double-sided differentiation where deployed at no increase in the elapsed time of execution. Similarly, rotorcraft configurations that required a higher number of perturbed states i.e. tilt-rotor aircraft - which have more control states than conventional main and tail rotor helicopters, would be able to utilise a larger number of processors and thus run the code without any increase in the elapsed time of execution.

2.3 Free Response

The RASCAL model can also be used to develop time histories of the vehicle states by calculating the response of the aircraft. This is useful as it allows evaluation of the aircraft's response to a control perturbation and/or gust.

As no 'control state' loop exists the only parallel implementation possible is the time-step implementation. The model has been parallelised using the exact inertial accelerations and also with remodelled ones in order to remove the necessity of inverting the matrix A.

2.3.1 Time-step Parallelisation

2.3.1.1 Description

The model was parallelised in a similar manner to that describe in section 2.1.1.1.

2.3.1.2 Results

The case was 10 seconds of PUMA flight time. The parallel performance improvement is shown in table 5.

	T800	R4400
1 Node	23897s	258s
10 Nodes	10470s	264s
Speed-up	2.28	0.98

Table 5

2.3.1.3 Discussion

Similar conclusions to those in section 2.1.1.3 can be drawn.

2.3.2 Time-step Parallelisation - Remodelled accelerations

2.3.2.1 Description

In order to try to improve the parallel performance of the code the inertial accelerations were remodelled in order to remove the necessity of the inversion of the matrix A. This remodelling is described fully in Appendix 2.

2.3.2.2 Results

Due to the high degree of communication in this application it has only been implemented on the Parsytec machine.

The results are given in table 6.

	T800
1 Node	6855
10 Nodes	1054
Speed-up	6.50

Table 6

2.3.2.3 Discussion

The remodelling of the inertial accelerations has a major effect on the parallel performance of the code.

The parallel performance improvement is very good. It should also be noted that on a single T800 node the sequential performance is also dramatically improved.

It would also be possible to achieve 'double-implementations' with greater speed-up in the calculation of trim and of small perturbation.

3. Conclusions

1. The rotorcraft simulation model RASCAL has been successfully parallelised in a number of different ways.

2. Whilst the parallel implementation has only been implemented for the main and tail rotor helicopter all of the message passing routines have been written so that they can be easily extended in include the full suite of rotorcraft that the RASCAL model is able to represent.

2. The 'coarse-grained' implementations display excellent performance improvement on both the parallel computer and the clustered workstations. The 'fine-grained' implementations only display useful improvement on the parallel computer.

3. The 'coarse-grained' implementations display better performance improvement than the 'fine-grained' ones.

4. The user-transparency has where possible been maintained. For the time-step parallelisation where much message passing is necessary the flexibility in the number of processors has been removed in order to maintain the generic configurational qualities of the simulation.

5. The developer-transparency has been maintain and all of the parallel elements of the code are contain within distinct sub-routines. This allows straightforward conversion to any message passing language and also keeps the main sequential body of code intact, thus allowing it to be developed independently of the message passing routines.

6. The mathematical modelling can have a significant effect on the ability to parallelise the code effectively. For example, disc type wake models require summed forces and moments which would not be required for an individual blade type wake model - so the amount of message passing required is increased. Also the modelling of the blades inertial forces and moments has a dramatic effect on the sequential run-time performance of the code and also its ability to be parallelised. It is thus necessary to carefully consider the impact of the modelling used on the parallelism of the code if run-time performance is of concern.

7. Parallel computing can only be a useful tool when the model that is implemented is intrinsically parallelisable. Rotorcraft simulation does display enough inherent parallelism to make parallel processing technology a valid tool.

4. Further Work

1. Development of a parallel trim algorithm which captures the periodicity of the aircraft's trim state rather than a 'quasi-steady' approximation. This can be implemented by checking the periodicity of the full state vector of the vehicle and ensuring that its mean flight path is the required one. There exists vast scope for parallelisation in the GENISA algorithm here.

2. Incorporation of different wake models which are not based on viewing the rotor system as a disc thus reducing the message passing required.

3. Code optimisation to achieve real-time simulation.

Figures

Figure 1 - Time-step Parallelisation

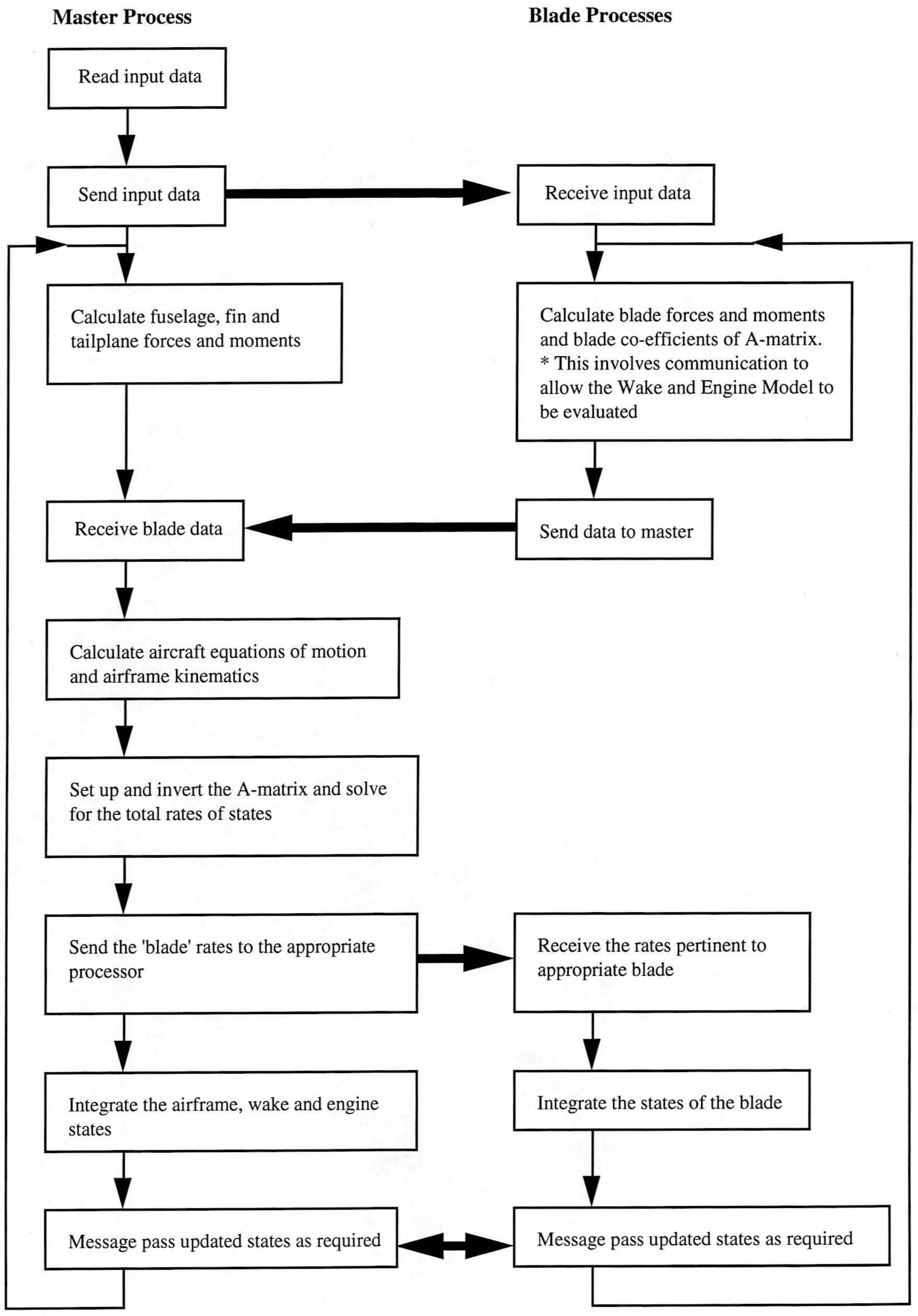

Master Process
Blade Processes
Read input data
Send input data
Receive input data
Calculate fuselage, fin and tailplane forces and moments
Calculate blade forces and moments and blade co-efficients of A-matrix. * This involves communication to allow the Wake and Engine Model to be evaluated
Receive blade data
Send data to master
Calculate aircraft equations of motion and airframe kinematics
Set up and invert the A-matrix and solve for the total rates of states
Send the 'blade' rates to the appropriate processor
Receive the rates pertinent to appropriate blade
Integrate the airframe, wake and engine states
Integrate the states of the blade
Message pass updated states as required
Message pass updated states as required

Figure 2 - Trim Algorithm Parallelisation

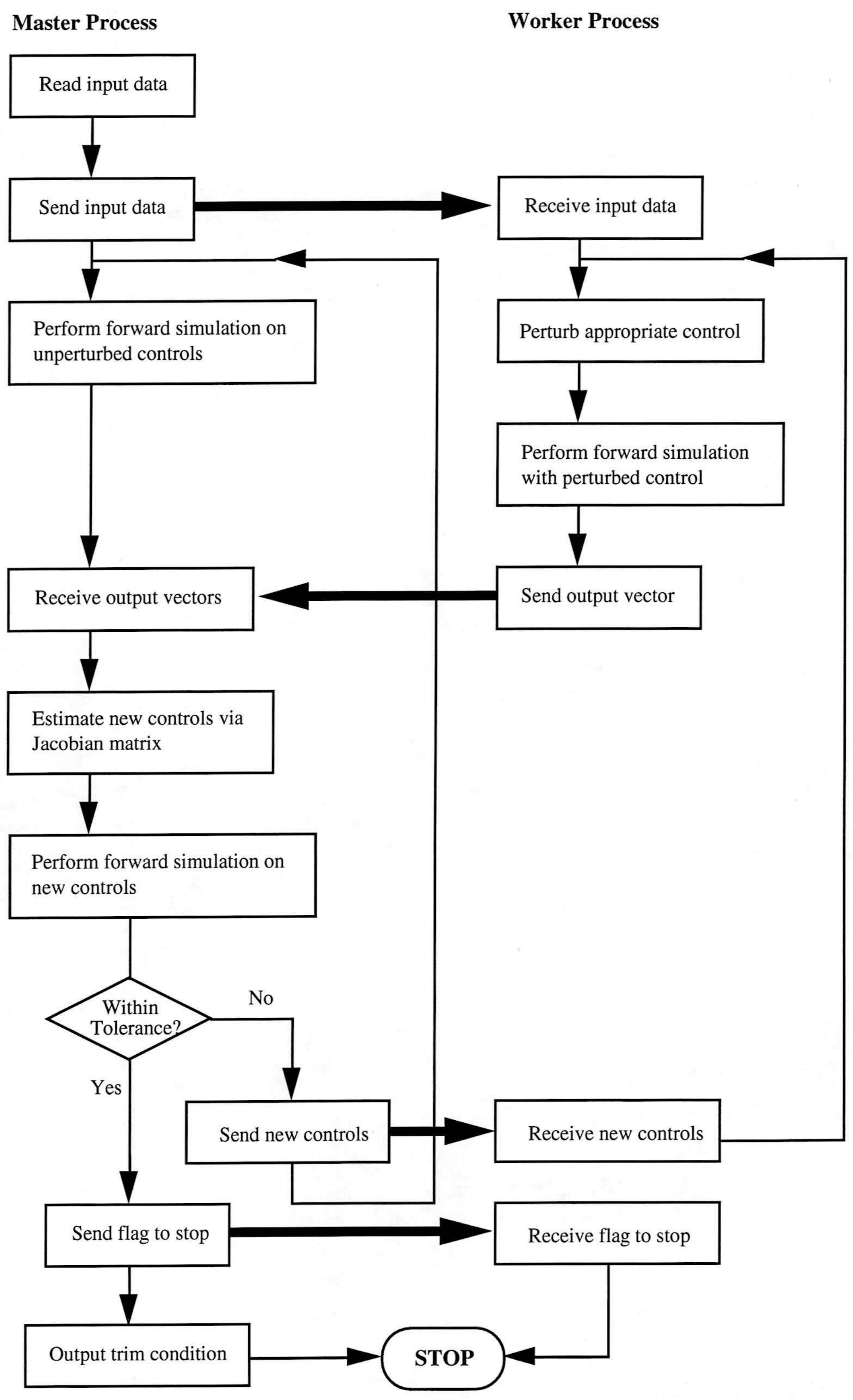
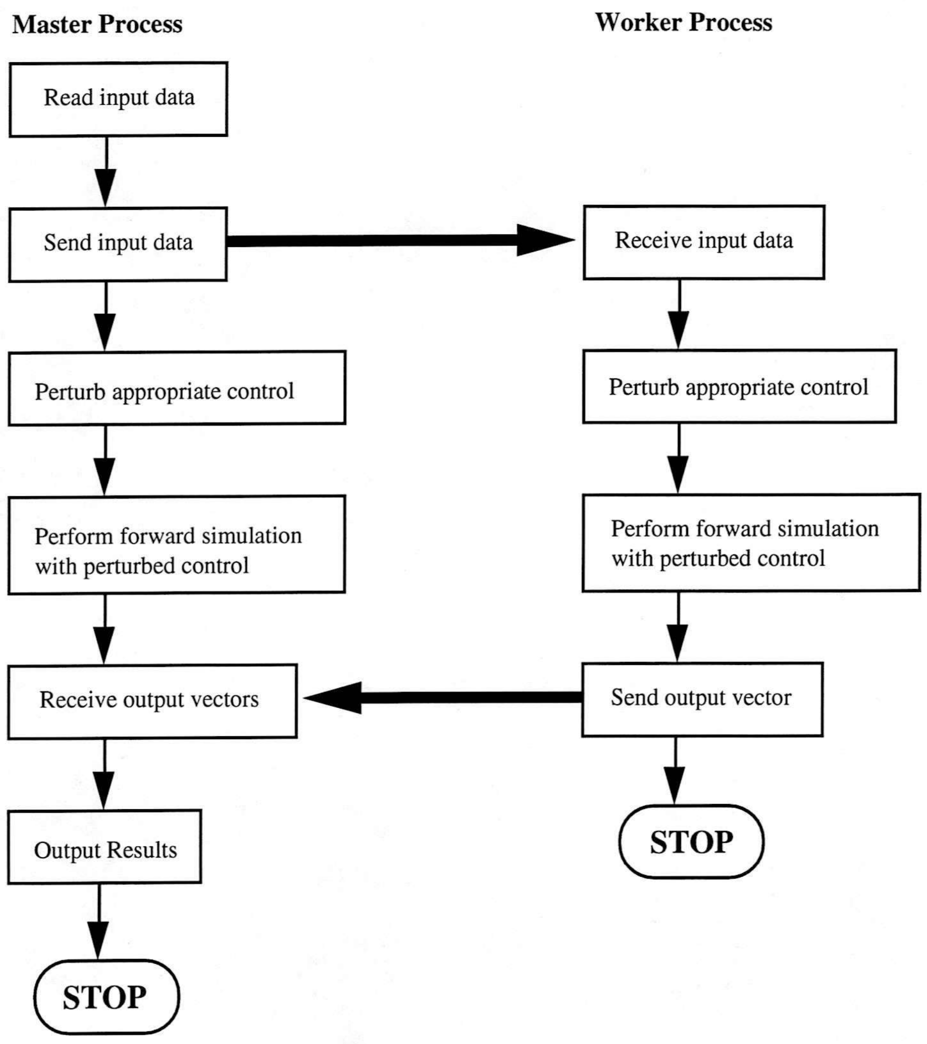


Figure 3 - Perturbation Parallelisation


Master Process
Worker Process
Read input data
Send input data
Receive input data
Perturb appropriate control
Perturb appropriate control
Perform forward simulation with perturbed control
Perform forward simulation with perturbed control
Receive output vectors
Send output vector
Output Results
STOP
STOP

Appendices

Appendix 1.

Common PARIX functions

myprocid()	- returns a unique processor identification number.
nprocs()	- returns the number of processors within the partition.
addnewlink()	- creates a virtual link between two processors in the partition.
send()	- sends information via a virtual link.
recv()	- receives information via a virtual link.

Common PVM functions

pvmfmytid()	- returns Unix process identification number
pvmfspawn()	- starts 'child' task
pvmfparent()	- returns Unix process identification number of 'parent' process
pvmfinit()	- clears memory in buffer
pvmfpack()	- packs data into buffer memory
pvmfsend()	- sends buffer
pvmfrecv()	- receives buffer
pvmfunpack()	- unpacks data from buffer

Appendix 2.

Calculation of the acceleration of a blade element

From ref 8, it is shown that the absolute acceleration, $\underline{a}_p$, of a point moving in a co-ordinate system with rotating axis is given by:

$$\underline{a}_p = \underline{a}_o + \frac{d^2 \underline{r}}{dt^2}$$

where

$$\frac{d^2 \underline{r}}{dt^2} = \underline{a}_{rel} + 2\underline{\omega} \times \underline{u}_{rel} + \underline{\omega} \times (\underline{\omega} \times \underline{r}) + \underline{\dot{\omega}} \times \underline{r}$$

As there is no motion between the points that we are considering and that the rotorcraft components in question are modelled as rigid bodies, we can say that:

$$\underline{a}_{rel} = \underline{u}_{rel} = 0$$

Thus

$$\frac{d^2 \underline{r}}{dt^2} = \underline{\omega} \times (\underline{\omega} \times \underline{r}) + \underline{\dot{\omega}} \times \underline{r}$$

So in notation that is consistent with RASCAL we can say

$$\underline{\dot{u}}_{hub}^{body} = \underline{a}_{cg}^{body} + \underline{\omega}_{cg}^{body} \times (\underline{\omega}_{cg}^{body} \times \underline{r}_{hub/cg}) + \underline{\dot{\omega}}_{cg}^{body} \times \underline{r}_{hub/cg}$$

where

$$\underline{a}_{cg}^{body} = \underline{\dot{u}}_{cg}^{body} + \underline{\omega}_{cg}^{body} \times \underline{u}_{cg}^{body}$$

It is now possible to transform the axis set to one which is located in the rotorcraft hub by the following transformation.

$$\underline{\dot{u}}_{hub}^{shaft} = T_1 \underline{\dot{u}}_{hub}^{body}$$

where

$$T_1 = \begin{bmatrix} \cos\theta_s & 0 & -\sin\theta_s \\ \sin\phi_s \sin\theta_s & \cos\phi_s & \sin\phi_s \cos\theta_s \\ \cos\phi_s \sin\theta_s & -\sin\phi_s & \cos\theta_s \cos\theta_s \end{bmatrix}$$

Transforming the axis set again from one which is located in the hub to one which is rotating with the hub can be achieved as follows:

$$\underline{\dot{u}}_{hub}^{rotor} = T_2 \underline{\dot{u}}_{hub}^{shaft}$$

where

$$T_2 = \begin{bmatrix} \sin\psi & -\cos\psi & 0 \\ \cos\psi & \sin\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

So the acceleration can now be calculated at the hinge using the following transformation:

$$\underline{\dot{u}}_{hinge}^{rotor} = \underline{\dot{u}}_{hub}^{rotor} + \underline{\omega}_{hub}^{rotor} \times (\underline{\omega}_{hub}^{rotor} \times \underline{r}_{hinge/hub}) + \underline{\dot{\omega}}_{hub}^{rotor} \times \underline{r}_{hinge/hub}$$

And this can be further transformed into blade axes by:

$$\underline{\dot{u}}_{hinge}^{blade} = T_3 \underline{\dot{u}}_{hinge}^{rotor}$$

where

$$T_3 = \begin{bmatrix} \cos\zeta & \sin\zeta & 0 \\ -\cos\beta\sin\zeta & \cos\beta\cos\zeta & \sin\beta \\ \sin\beta\sin\zeta & -\sin\beta\cos\zeta & \cos\beta \end{bmatrix}$$

Finally, the acceleration of a blade element can be calculated, using:

$$\underline{\dot{u}}_{elem}^{blade} = \underline{\dot{u}}_{hinge}^{blade} + \underline{\omega}_{hinge}^{blade} \times (\underline{\omega}_{hinge}^{blade} \times \underline{r}_{elem/hinge}) + \underline{\dot{\omega}}_{hinge}^{blade} \times \underline{r}_{elem/hinge}$$

It is obvious from the above that the angular velocities' and accelerations at each intermediate point is required in order for these transformations to take place. Consider first the angular velocity.

$$\underline{\omega}_{hub}^{body} = \underline{\omega}_{cg}^{body} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

Transforming this into axes fixed in the hub can be achieved as follows:

$$\underline{\omega}_{hub}^{shaft} = T_1 \underline{\omega}_{hub}^{body}$$

And into and axes system that rotates with the shaft

$$\underline{\omega}_{hub}^{rotor} = T_2 \underline{\omega}_{hub}^{shaft} + \begin{bmatrix} 0 \\ 0 \\ \Omega \end{bmatrix}$$

We can also say that

$$\underline{\omega}_{hinge}^{rotor} = \underline{\omega}_{hub}^{rotor}$$

And we can transform this into blade axes using the following transformation:

$$\underline{\omega}_{hinge}^{blade} = T_3\underline{\omega}_{hinge}^{rotor} + \begin{bmatrix} \dot{\beta} \\ \dot{\zeta} \\ \dot{\theta} \end{bmatrix}$$

The same can also be done for angular acceleration:

$$\underline{\dot{\omega}}_{hub}^{body} = \underline{\dot{\omega}}_{cg}^{body} = \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix}$$

Which can be transformed into an acceleration in shaft axes:

$$\underline{\dot{\omega}}_{hub}^{shaft} = T_1\underline{\dot{\omega}}_{hub}^{body}$$

And from that into one which rotates with the hub

$$\underline{\dot{\omega}}_{hub}^{rotor} = T_2\underline{\dot{\omega}}_{hub}^{shaft} + \begin{bmatrix} 0 \\ 0 \\ \dot{\Omega} \end{bmatrix}$$

It is therefore apparent that:

$$\underline{\dot{\omega}}_{hinge}^{rotor} = \underline{\dot{\omega}}_{hub}^{rotor}$$

So it is now possible to describe the angular acceleration as in blade axes as follows:

$$\underline{\dot{\omega}}_{hinge}^{blade} = T_3\underline{\dot{\omega}}_{hinge}^{rotor} + \begin{bmatrix} \ddot{\beta} \\ \ddot{\zeta} \\ \ddot{\theta} \end{bmatrix}$$

From the description above it is apparent that the calculation of the inertial acceleration of a blade element is a function of many parameters including translational acceleration of the vehicle, angular acceleration of the vehicle, rate of change of rotorspeed and angular acceleration of the blades. This complicates the mathematics of the simulation somewhat as the model works by calculation of the rate of change of each state variable and thus the new state is found by numerical integration.

Mathematically this can be expressed as follows:

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

In RASCAL this is overcome in the following manner:

$$\underline{\dot{x}}_{total} = \underline{\dot{x}}_{partial} + A'\,\underline{\dot{x}}_{total}$$

Which can be re-arranged to give the following:

$$\underline{\dot{x}}_{total} = A^{-1}\underline{\dot{x}}_{partial}$$

Where

$$A = I - A'$$

This allows the inertial acceleration of each blade element to be calculated exactly but causes the computational performance to be poor as it involves the calculation of many extra terms and also requires the matrix A to be inverted (or for the system of differential equations to be solved for directly).

It is possible to avoid this computational burden by introducing a modelling discrepancy. That is to use information from the previous time step when calculating the blade element inertial accelerations. It is possible to do this without introducing large errors because the contribution of the inertial forces and moments to the overall forces and moments of the rotor system is small because of the dominance of the aerodynamic terms and also if the model is calculated over small time steps then variations in the rates of change will also be small.

References

1. Padfield, G.D., 'Helicopter handling qualities and control - Is the community ready for change?'. Proceedings of the RAeS Conference on Helicopter Handling Qualities and Control, London (1988). ISBN 0 903409 24 0

2. Viehover, G. et al, 'Parix Software Documentation'. Aachen (1993)

3. Geist, A. et al, 'PVM: Parallel Virtual Machine'. MIT Press (1994)

4. Houston, S.S., 'Rotorcraft Aeromechanics Simulation for Control Analysis - Mathematical Model Description'. Internal Report No. 9123 (1991)

5. Houston, S.S., 'Validation of a non-linear individual blade rotorcraft flight dynamics model using a perturbation method'. The Aeronautical Journal of the Royal Aeronautical Society (1994)

6. Rutherford, S., Thomson, D.G., 'Development of a Generic Inverse Simulation Algorithm'. Internal Report No. 9410 (1994)

7. McVicar, J.S.G., 'A Generic Tilt-Rotor Simulation Model with Parallel Implementation', Ph.D Thesis (1993)

8. Meriam, J.L., Kraige, L.G., 'Engineering Mechanics - Volume 2 Dynamics', John Wiley and Sons Inc.(1993)