MULTIPROCESSOR SIMULATION OF THE DYNAMICS OF SPACE STRUCTURES

Socrates Dimitriadis

September 1986
CSD-860030

UNIVERSITY OF CALIFORNIA

Los Angeles

Multiprocessor Simulation of the Dynamics of Space Structures

A dissertation submitted in partial satisfaction of the

requirements for the degree Doctor of Philosophy

in Computer Science

by

Socrates Dimitriadis

1986

The dissertation of Socrates Dimitriadis is approved.

[illegible]

Stephen E. Jacobsen

[illegible]

James B. Ralston

Milos Ercegovac

Milos D. Ercegovac

Jack W. Carlyle

Jack W. Carlyle

[illegible]

Walter J. Karplus, Committee Chair

University of California, Los Angeles

1986

DEDICATION

To those that prepare the humanity for the new era

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

First I wish to thank the members of my doctoral Commitee, Professors Jack Carlyle, Milos Ercegovac, Stephen Jacobsen, Walter Karplus (chairman) and James Ralston. I particularly thank Professors Walter Karplus and Jack Carlyle, as well as Dr. Rafael Gluck, member of the TRW Space and Technology Group, for their scientific advice and cooperation during this research study.

In addition, I wish to thank Professor Walter J. Karplus, Dr. Rafael Gluck, the two Greek Institutions, "Alexandros Onassis Foundation" and "Bodosakis Foundation", as well as the Phi Beta Kappa Association for appreciating my scientific skills and providing me with all the needed finacial support to acomplish my graduate studies.

I owe a great deal of gratitude to Professor Walter Karplus who has been far more than an academic advisor to me. He has been unusually patient honest and friendly, appreciated my ideas and plans for the future and tried to help me to achieve them.

I also particularly thank Professor Jack Carlyle for being very supportive, and for providing me with a lot of technical information that will hopefully be very helpful to me to achieve my future plans. I would also like to thank Professor Stephen Jacobsen for accepting to serve on my committee on short notice.

I would also like to thank Verra Morgan, who was very informative and offered me various kinds of assistance that have eased my life as a foreign student. Finally, I wish to thank my beloved Evangelia Kardara for her love and patience, as well as my new friends Shun Cheung, Jose Arabe, T.M. Ravi and Ljiliana Trajkovic

for their friendship and for keeping my life at UCLA interesting by providing me with information about the different parts of the world where they originate from.

VITA

February 5, 1959	Born, Gipsochorio Pellas, Greece
1976-1981	MS in Electrical Engineering, Aristotelian University of Thessaloniki, Greece
1981-1983	MS in Computer Science, University of California Los Angeles
1983-1986	PhD in Computer Science, University of California Los Angeles

1. INTRODUCTION

The objective of the dissertation is the development of a set of tools to optimize the use of multiprocessor computer systems in the simulation of the dynamics of space structures. For the better understanding of the scope of the dissertation, the requirements of the simulation of the dynamics of space structures are presented in the first section. Section 2 presents the particular multiprocessor computer architectures that are considered. Then, important software with which this computers have to be equipped is discussed in Section 3. The objective of the dissertation and the approach are discussed in Section 4, followed by a presentation of the contributions of the dissertation in Section 5. An outline of the dissertation is given at the end of the chapter, in Section 6.

1.1 Dynamic simulation of space structures

There has been a great interest in the recent years in a new generation of space systems, space transportation systems, such as the space shuttles and space stations. These vehicles contain large multibodied flexible mechanical structures. The structure of a typical space shuttle is shown in Figure 1.1. The dynamic response of space structures to any type of excitations is in general characterized by oscillations. In the case of the space shuttle of Figure 1.1 for example, the application by the operator of a force on one of the arms to change the orientation of the array that it holds, would cause this arm and its array to oscillate until they reach the desired steady state position. Due to their physical coupling with the arm, other bodies will be forced to follow these oscillations. The oscillations contain high frequencies due to the flexibility of some of the bodies. In addition, they are slowly damped due to the lack of air friction in space.

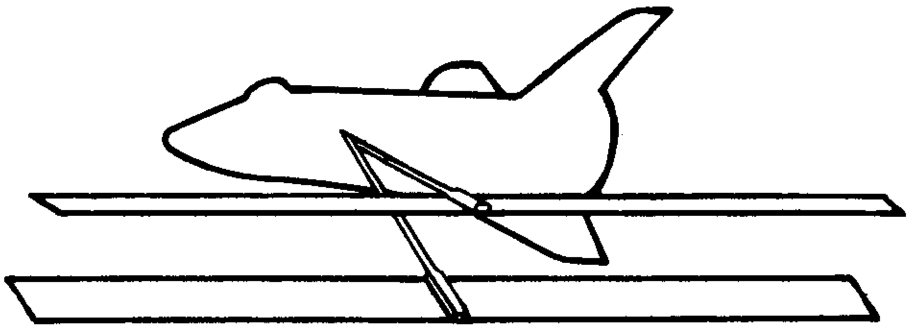


Figure 1.1: The structure of a space shuttle.

The optimization of the design of the bodies of the structure and of the systems to control their oscillations requires the estimation of several of their characteristics, such as frequency content, amplitude and damping rate. The cost of creating space conditions prohibits the measurement of these characteristics by experimenting with real vehicles. Instead computer simulations must be performed so that the dynamic response of the space structure is computed from its mathematical model.

The need to advance the state of the art in these simulations is apparent in the space industry. Aerospace engineers require tools that could permit high speed and low cost simulations. The hardware and the architecture of modern computer systems has reached the speed required by such simulations. However, sufficiently powerful software that fully exploits this computer power has not as yet been developed. Moreover, most of the existing high speed computers are too expensive.

The need for higher speed and lower cost in the simulations is due to the complexity of the mathematical models of the dynamics of the space structures. To appreciate the complexity of the models the reader is suggested to refer to the widely accepted DISCOS model [Bodl78], available commercially as a program. The largest part of the computation involved in the manipulation of such mathematical models by a computer, is the solution of a system of ordinary differential equations (ODEs) with very high computational requirements. This ODE system expresses the acceleration of the oscillating motion of the bodies and of the hinges that connect them. The solution of this ODE system provides the velocities and displacements of their oscillations, which in turn can provide any other information needed.

The solution of these ODE systems is time consuming because they are:

1. Large, consisted of hundreds of ODEs.
2. Highly oscillating and lightly damped.
3. Highly nonlinear including discontinuities.

The computational requirements are increased additionally by the need for high speed and high accuracy solutions. In particular, the transient solution of the ODE system has to be obtained in detail and accurately. The high frequencies of the oscillations of the transient and the length of the transient due to the low damping of the oscillations, require that a large number of solution points be computed. The accurate computation of these points is a difficult task.

1.2 The multiprocessor computer architecture

With today's technology single processor computers are not capable in general to handle computations of size and complexity of the order of our simulations. Experiments with computers with a single very powerful processor, reported in [Gluc85], show their inadequacy to achieve high speed simulation of the dynamics of the space shuttle shown in Figure 1.1. These experiments included among others the CRAY 1S, the Cyber 205 and large IBM mainframes, all employing a very powerful general purpose processor. Even computer systems that employ a special purpose processor as a peripheral device attached to a minicomputer host, called peripheral array processor (PAP) computer systems, are shown to be inadequate. Two of most powerful PAPs, the AD 10 and the FPS 164, were included in the experimental comparison. The need for multiprocessor architecture computers is apparent.

The multiprocessor computers can be classified as:

1. Shared memory distributed systems

2. Message passing distributed systems

Shared memory computers are tightly coupled distributed systems that employ a number of parallel processors that share a common multiport memory. The processors access the modules of the memory through an interconnection network. The CRAY XMP computer is a well known example of shared memory system. The shared memory systems are more efficient for computations which permit data "transfer" between the processors without causing conflicts in accessing the common memory. If there is a need to transfer the same data from one processor to all or some of the others, memory access conflicts would occur. The resolution of the memory access conflicts causes an overhead that reduces the speed of the simulation. In Chapter 6 it is shown that the solution of the ODE systems in our simulation problems requires periodically some processors to transfer their results to several of the other processors at the same time. This means that several processors have to access the same data on a processor simultaneously resulting in a conflict. The computation involved however, consists of long loosely coupled processes. As a result, the interprocessor communication is relatively infrequent, and the memory access conflict resolutions overhead would not be significant.

Message passing computers are either loosely coupled or tightly coupled distributed systems that employ a number of processors with their own memory that are interconnected via a network. Loosely coupled systems use simple inexpensive communication networks, such a small number of busses. They are preferable for computations with fairly independent processes, such as ours. The Cm* is a well known example of loosely coupled system. Similar to the memory access conflicts for shared memory systems, bus access conflicts cause an overhead in loosely coupled systems. Again, the relatively infrequent communication required would not cause a significant overhead. Tightly coupled systems use more sophisticated and expensive communication networks to reduce the communication overhead. They are more appropriate for computations with highly dependent processes. A network of moderate sophistication would be an appropriate compromise between speed and cost. The CAPPS system [Gluc85] developed by Paragon Pacific is an interesting example based on this compromise. It is discussed briefly in Appendix A.

The conclusion of the above discussion is that both shared memory and message passing multiprocessor computers can be appropriate for our computations. The studies in this dissertation are applicable to both architectures. Abstract models of the two architectures that are adapted are shown in Figure 1.2 and 1.3. It should be realized that in addition to memory or bus access overhead, both architectures encounter another overhead due to the execution of the protocols of communication. A drastic reduction of this overhead can be achieved by using a input/output (I/O) system that operates relatively independently from the other operations within the processors. The execution of I/O operations of the processors can then be overlapped to a great extend with the arithmetic or other operations of the processors. The models in Figure 1.2 and 1.3 take this approach.

1.3 Software for the simulation process

The simulation process consists of:

1. The mathematical modeling of the dynamics of the space structure.

2. The development, from the mathematical model, of the application program to be executed by each processor of the computer.

3. The translation of the application program by the system software of the computer, to machine code executable by the hardware of each of the processors.

4. The execution of the machine code by the processors that produces the simulation results.

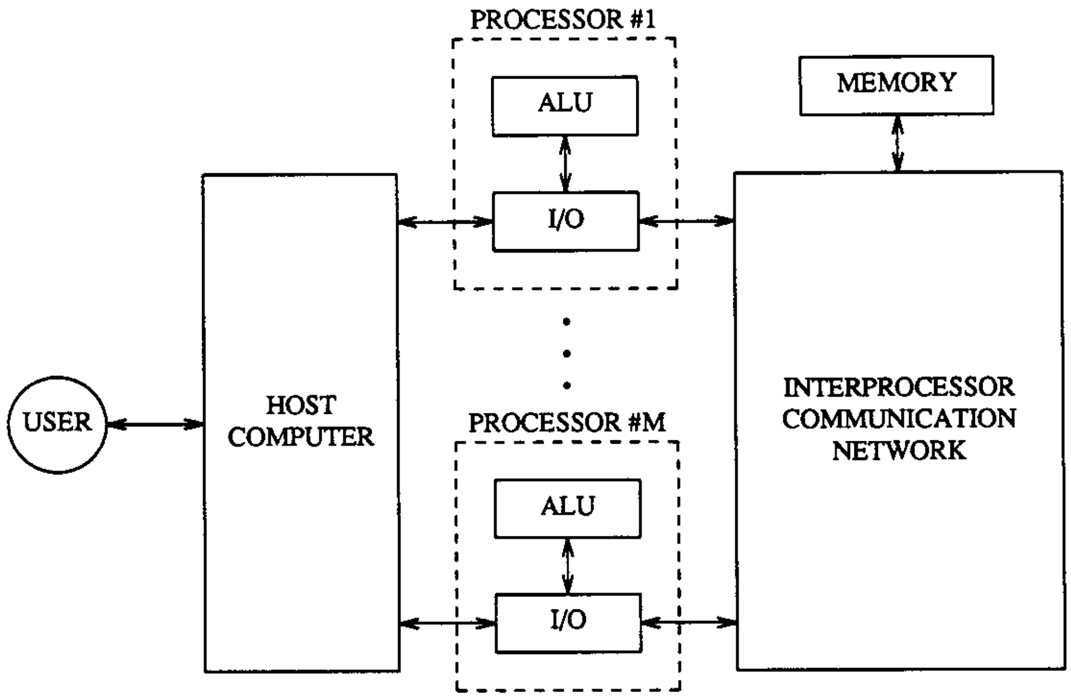


Figure 1.2: Abstract model of a shared memory multiprocessor computer

The efficiency of a simulation for a specific computer is primarily determined by the simplicity and accuracy of the mathematical model and by the speed and accuracy of the numerical algorithm used to compute the dynamic response of the space structure from its model. Benchmarks have shown clearly that currently available computers cannot efficiently handle detailed models. Therefore the design analysts must provide simplified models to the computer. These simplifications entail the elimination of those terms in the model that have a negligible effect on the solution, so that the accuracy of the simulation is not reduced excessively.

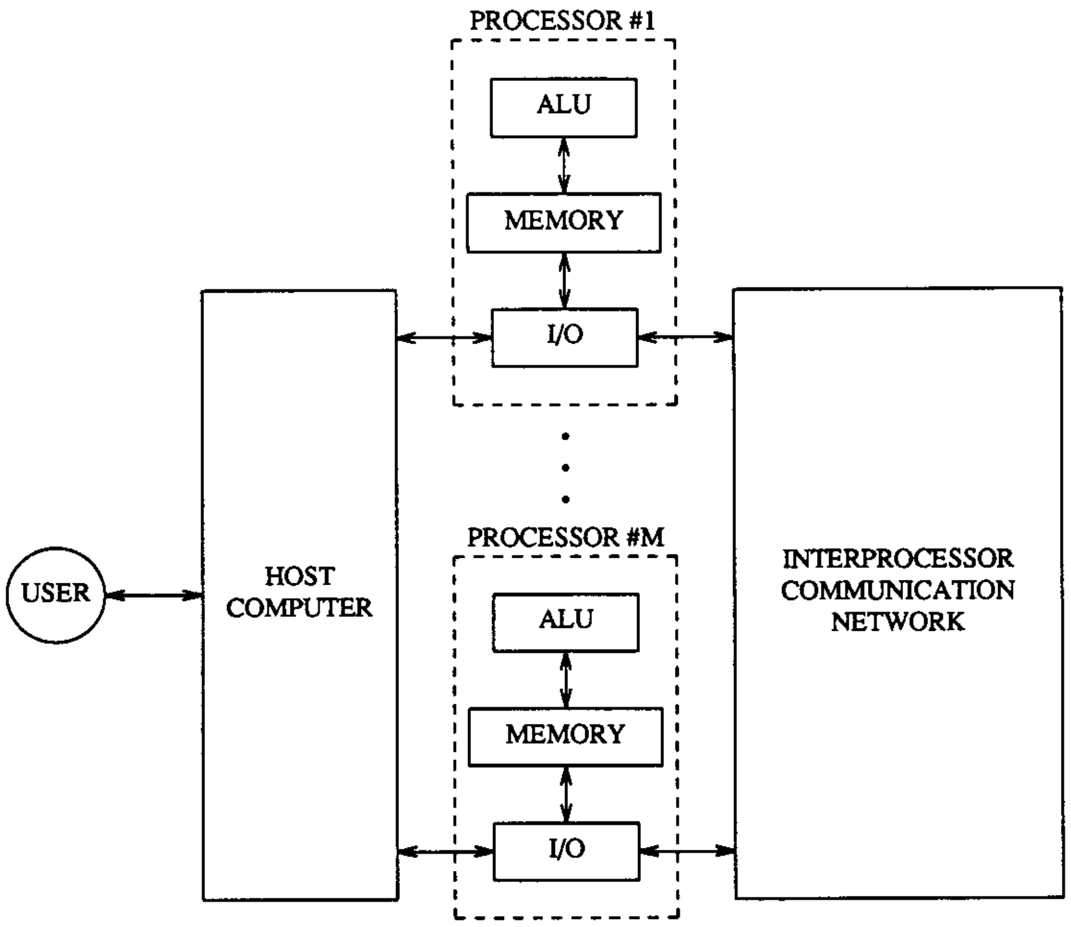


Figure 1.3: Abstract model of a message passing multiprocessor computer

Straightforward modeling results in an implicit formulation of the ODE system, as $\Phi(\dot{x},x,u,t)=0$, where the state variables x(t) and the input variables u(t) represent the velocities and the excitations of the structure. This is the only feasible analytic modeling for large and complex problems like ours. The model DISCOS is an implicit one. Unfortunately, simplifications on an implicit ODE system are practically impossible for large systems. Terms that could be eliminated are not easily identifiable. Therefore the analyst has no direct control over the complexity of his model. On the other hand, explicit formulation of the ODEs, i.e. formulation as state equations $\dot{x}=F(x,u,t)$, permits simplifications. Unfortunately, the size and complexity of our simulations prohibits explicit formulation analytically. The task of producing an explicit model from an implicit model has to be left to the computer. Symbolic manipulation programs are now available that can perform such a task. The SMP program [Cole81], an advanced version of MACSYMA, can reformulate an implicit ODE system $\Phi(\dot{x},x,u,t)=0$ as $A(x,t)\dot{x}=b(x,u,t)$. This is not exactly an explicit formulation, but one that can be termed semiexplicit. It has however, the characteristics of an explicit formulation that terms that can be eliminated are easy to identify. The semiexplicit formulation is obtained by gathering together in matrix form the acceleration terms on the left hand side and the rest of the terms on the right hand side.

The simulation process can be depicted as shown in Figure 1.4. This is not the only possible approach but it seems to be the most efficient. Other approaches skip the semiexplicit formulation and develop the application software based on numerical techniques to solve implicit ODE systems. The inefficiency of the implicit model however then limits the performance of the simulation.

1.4 The objective and approach of the dissertation

Although sufficient techniques and software exist for the modeling and the translation tasks, this cannot be claimed for the most important part, the application software generation. This task is heavily dependent upon the particular hardware. Each computer, or limited class of computers, requires an individual application software generator. However, concepts and techniques can be developed, that can

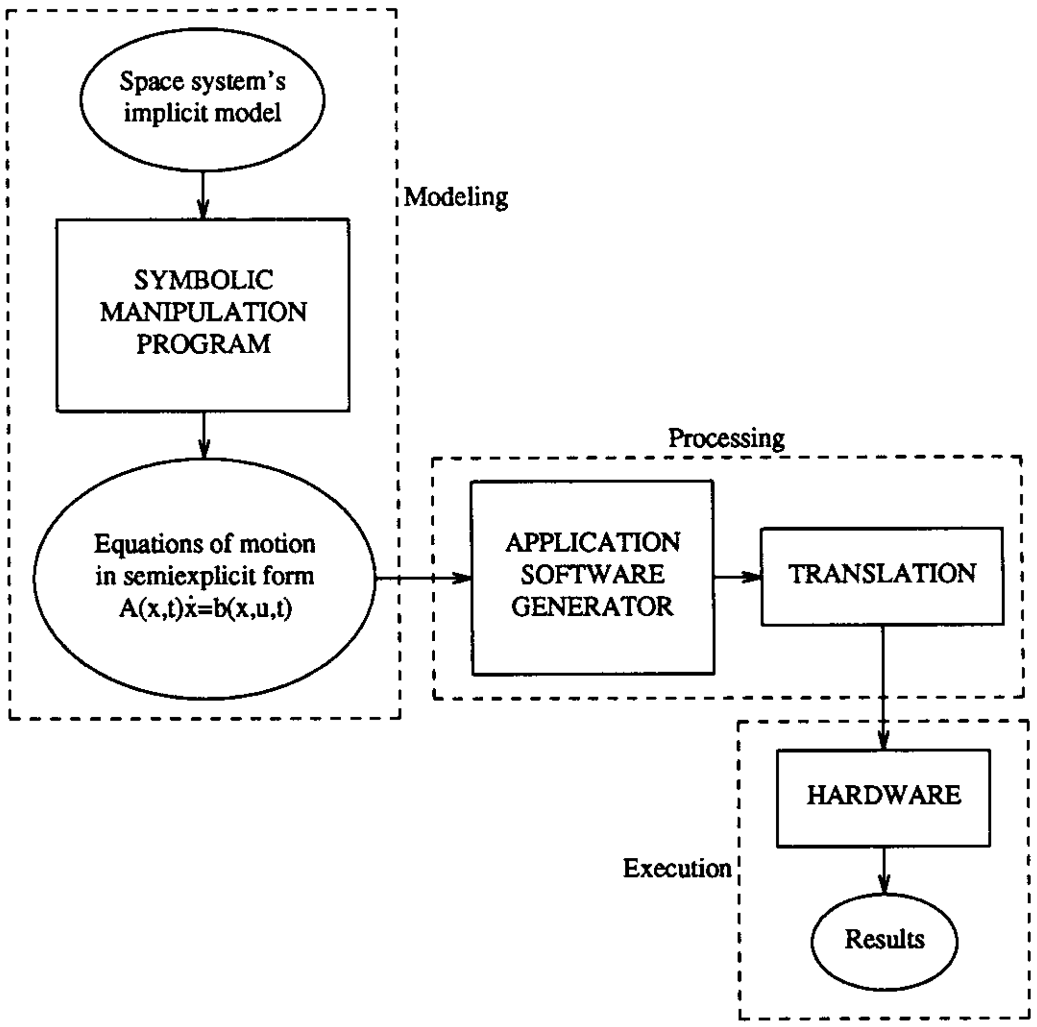

Space system's implicit model
SYMBOLIC MANIPULATION PROGRAM
Equations of motion in semiexplicit form $A(x,t)\dot{x}=b(x,u,t)$
Modeling
Processing
APPLICATION SOFTWARE GENERATOR
TRANSLATION
HARDWARE
Execution
Results

Figure 1.4: The simulation process using a symbolic manipulation program

apply to the generation of application software for any multiprocessor computer. The objective of this dissertation is the development of tools that facilitate the design and development of the application software generator for any multiprocessor computer system with an architecture such as the one shown in Figures 1.2 or 1.3. The criteria for the development of the design tools are the criteria for the efficiency of the application software generator. These criteria should be related to the ultimate goal of the maximization of the speed and of the minimization of the cost of the entire simulation process.

The speed of a simulation depends primarily on the speed of the solution of the ODE system involved. The solution speed in turn is primarily dependent on the efficiency of the model and of application software. Provided that the model has been optimized, the primary design objective of the application software generation is the maximization of the speed of the solution. This maximization is constrained by the required accuracy of the solution and by the hardware. Minimum hardware, i.e. minimum number of processors, is required to minimize the simulation cost. The maximization of the speed itself is an important contribution to the minimization of the cost as well.

A significant reduction of the simulation cost can be achieved by having a simple and modular enough application software generator. A simple and modular generator has several desirable features. It is easy to implement, to debug and to modify. In addition, it makes the prediction of performance easier. These are important contributions to the minimization of the simulation cost. The speed of the generator is not crucial because the application software can be, and usually is, generated in advance of the execution of the simulation. The translation of the application program to machine code is also performed in advance. The simulation time is therefore reduced to the time of the actual execution of the machine code. The time spent in the automatic generation of the application software however, is desirable to be short in order to minimize the entire simulation cost. A simple generator would, in most cases, also be faster.

The criteria for the development of the design tools are:

1. Maximization of the speed of the solution of the ODE system: The application software to be generated should provide a fast enough solution.

2. Simplicity of the tools: The generator, and therefore the tools for its development, should be simple and should provide the application software as rapidly as possible.

The solution speed in particular, can be maximized by:

1. The minimization of the computational load.

2. The optimization of the distribution of the minimized computational load among the processors.

The computational load depends primarily upon the numerical algorithm that is used to solve the ODE system. Therefore, the minimization of the computational load can be achieved by an appropriate selection of the algorithm. The appropriate implementation of the algorithm on the multiprocessor architecture optimizes the computational load distribution. The design tools that are developed in this dissertation include:

1. A procedure for the selection of the appropriate numerical algorithm to solve the system of ODEs that appears in the mathematical model.

2. A procedure for the multiprocessor implementation of the algorithm on a particular computer.

3. A procedure for the prediction of the performance of the particular computer.

The procedure for algorithm selection is based on the performance evaluation of a set of candidate algorithms. It accepts as inputs a description of the ODE system and of the candidate algorithms, as well as the bounds on the performance, and performs a:

1. Specification of the optimal characteristics of each algorithm.

2. Assignment of a relative value of performance to every algorithm with its optimal characteristics.

3. Selection of the algorithm with the highest performance.

The procedure for the multiprocessor implementation of the selected algorithm accepts as inputs a description of the ODE system, of the algorithm to be used and of the computer architecture, as well as the performance bounds, and performs a:

1. Selection of the number of processors.

2. Partitioning of the computation into a number of tasks.

3. Assignment of these tasks to the processors for execution.

4. Scheduling of the execution of the tasks on each processor.

The procedure for the prediction of the performance of the computer is a part of the procedure for the multiprocessor implementation of algorithms. It accepts as inputs the computer architecture and the results of the multiprocessor implementation of the algorithm, and provides timing information of the computation. This information includes the time to obtain one point of the solution and the efficiency of the utilization of the parallel processors. The performance prediction can be implemented with the help of any commercially available computer system simulator.

An analysis of the performance of the procedures for the selection of algorithms and for their multiprocessor implementation is included. This analysis is based on a set of benchmark ODE systems that are constructed synthetically to contain all the required characteristics, as well as on the architecture of the multiprocessor computer CAPPS. The verification of the procedures is an important byproduct of the performance analysis.

1.5 Contribution of the dissertation

The results of this dissertation aim to improve the state of the art of advanced simulations of dynamic structures, with particular emphasis on the structures of the modern space transportation systems. Its particular contribution is in the area of efficient use of multiprocessor computers to be used for such simulations. More general, the dissertation aims to be an important contribution to the field of distributed processing of ODEs. Although it focuses on the solution of systems of ODEs that appear in the dynamic simulation of space structures with computers of particular multiprocessor architectures, great effort is maid to keep the procedures applicable to a wide variety of cases. In general, the procedures and results of the dissertation provide useful indications for the approach to be taken in case of different multiprocessor architectures, of ODE systems with different characteristics and formulation, and of different algorithms.

As a result, the procedures and the results of this dissertation are useful in other dynamic structures applications including aerospace and robotics. The simulation of the dynamics of helicopters or the arms of robots for example, constitutes similar computational requirements and therefore can very well be approached in a way similar to the one proposed for space structures.

In addition to the development of the application software generator, the design tools can be useful starting points for:

1. The design of parallel numerical algorithms.

2. The design of the architecture of multiprocessor computers to be used for computations that involve the solution of ODE systems.

3. The design of system software, such as distributed compilers, for such multiprocessor computers.

1.6 Organization of the dissertation

The main purpose of this chapter is the presentation of the objective of the dissertation and the approach that will be taken to achieve this objective. Chapters 2, 3, 5 and 6 are devoted to the procedure for the selection of the algorithm. Chapter 2 introduces the procedure at a high level that is applicable to any type of ODE systems. Chapter 3 specifies the implementation of the procedure in the case of ODE systems that appear in the simulation of the dynamics of space structures. The selection of the candidate algorithms to solve this particular class of ODE systems is included. Chapter 6 evaluates the performance of the procedure when applied to benchmark ODE systems that are selected in Chapter 5.

Chapters 4, 5 and 7 are devoted to the procedure for the multiprocessor implementation of algorithms. Chapter 4 introduces the procedure and specifies ways for its implementation. It includes suggestions for the prediction of the performance of the computer. Chapter 7 evaluates the performance of the procedure when applied to benchmarks selected in Chapter 5. An important byproduct of this performance analysis is the comparison of the candidate algorithms for the simulation of space structures and the indication of the most advantageous of them.

Finally, Chapter 8 constitutes the conclusion of the dissertation. It includes a summary, a conclusion and suggestions for improvements and complementary future work.

Several Appendices appear at the end of the dissertation. These Appendices contain specialized information, that is more or less peripheral to the main objective of the dissertation, but could be of interest to some readers. Appendix A presents a description of the architecture of the CAPPS computer, an interesting message passing distributed system, which is selected in this dissertation to demonstrate and verify the proposed procedures. The benchmark ODE systems used for the performance analysis of the procedures are shown in Appendix B. Implementation of several modules of the multiprocessor implementation procedure, as FORTRAN subroutines, are presented in Appendix C. This appendix includes a subroutine that generates the benchmarks and subroutines that implement the two most interesting algorithms, the Explicit Power Series Expansion and the Adams Bashforth.

1.7 Summary

The design of the structure and the controls of the space systems, particularly of the recently developed space transportation systems, requires computer simulations of their dynamics. There is need to improve the speed and cost of these simulations, which require the solution of a large, nonlinear, highly oscillating and lightly damped system of ODEs. The long transient solution has to be obtained with high speed, high accuracy and low cost. Simplifications of the mathematical model of the ODE system are very important means to increase the speed of their solution. A semiexplicit formulation of the ODE system as $A(x,t)\dot{x}=b(x,u,t)$ has been proposed, because it permits the control of the complexity of the mathematical model. A semiexplicit ODE system can be derived automatically from the implicit formulation, which is easier to develop manually, using a symbolic manipulation program. The size and complexity of these ODE systems requires multiprocessor computers. The results of the dissertation are applicable to both message-passing and shared memory multiprocessor architectures.

The dissertation aims to improve the state of the art of the dynamic simulation of space structures. Its particular contribution is in the area of the efficient use of multiprocessor computers for such simulations. The objective of the dissertation is the development of a set of tools that facilitate the automatic generation of the application software for each processor. In particular, these tools include the selection of the appropriate algorithm to solve the ODE system, the multiprocessor implementation of this algorithm, and the prediction of the performance of the multiprocessor computer. The criteria for the development of the design tools include the maximization of the speed of the solution of the ODE system and the simplicity of the tools. These criteria are related to the maximization of the speed and the minimization of the cost of the simulations.

More general, the dissertation aims to be an important contribution to the field of distributed processing of ODEs. The design tools that are developed are general enough so as to be applicable to a wide variety of ODE systems, algorithms and multiprocessor computers. Therefore, the results of the dissertation can be useful to other engineering applications as well. The design tools offer a base for the development of parallel algorithms, and for the design of system software and hardware for multiprocessor computers to be used for similar applications.

2. SELECTION OF ALGORITHMS

As discussed in the introductory chapter, one of the objectives of the dissertation is the development of a procedure which can facilitate the automatic selection of an appropriate algorithm to solve ODE systems that appear in the mathematical models of the dynamics of space structures. The purpose of this and the following three chapters is to propose such a procedure. This chapter is devoted to the introduction of the procedure at a high level that is applicable to any type of ODE systems. The next chapter specifies the implementation of the high level components of the procedure in the case of ODE systems that appear in the models of the dynamics of space structures. Chapter 5 evaluates the performance of the procedure when applied on a benchmark ODE system that is proposed in Chapter 4. Here, the objective of the procedure is presented in the first section. Then, Section 2 introduces briefly the terms and issues in the numerical solution of ODEs that are related to the algorithm selection. This is followed by an approach to develop the procedure, suggested in Section 3. Section 4 is devoted to the presentation of the procedure itself. A brief informal validation of the procedure is given in the final Section 5.

2.1 The objective of the procedure

There has always been a great interest in the automation of the selection of algorithms for various classes of problems, including the solution of systems of ODEs. The selection of an algorithm to solve ODE systems is probably the most difficult, due to the usually large number of candidate algorithms for every ODE system. Despite of the great demand however, no software exists as yet to perform such a selection task. The reason probably is, that the problem is too wide and complicated to be solved in its generality. Different ODE systems may require a significantly different approach for selecting the appropriate algorithm for their solution. In addition, although the concepts and tradeoffs in selecting an algorithm are more or less understood, no complete and organized procedure that incorporates these concepts and tradeoffs, has as yet been suggested. As a result, analysts are forced to improvise every time they encounter such task.

The objective of this chapter is to propose a procedure based on an organized and quantitative strategy to select an algorithm. The procedure is modular enough so that its automation is feasible. It should be realized that the procedure aims to provide only a helpful starting point for the automation of the algorithm selection. The complete coverage of the details for different ODE systems and the actual implementation of the procedure is beyond the scope of this dissertation. A secondary purpose for developing the procedure is to investigate which algorithms might be the most advantageous to solve ODE systems that appear in the dynamic simulation of space structures. This investigation takes place in Chapter 5.

2.2 Numerical solution of systems of ODEs

Numerical algorithms work in a stepwise fashion. The entire solution is partitioned into a sequence of points. At every step, the algorithms compute the next point of the solution by using a recursive formula. The recursive formulations of the algorithms are approximate due to the finite number of terms that they incorporate. As a result, the points of the solution that the algorithms provide can only approximate the exact solution. The algorithms therefore operate like an approximate sampling of the exact solution.

An algorithm can be characterized by the order and step size of its formula. The order m denotes the number of terms in the formula. A different formula will result for different number of terms. It is apparent, that every algorithm has an infinite number of formulas. The step size h denotes the time distance between two consecutive points of the solution. In other words, the step size represents the inverse of the sampling rate of the solution. The selection of an algorithm means the selection of one of its formulas with particular order and step size.

For many ODEs, the use of constant order and step size would be preferable to variable ones. If variable order and step size are used for some ODE, a separate selection for each point of its solution must be made. This may result in a more optimal selection than the constant one. However, the decision process that has to be executed for each point of the solution can cause a significant overhead that could not be possible to compensate by the use of a more optimal order and step size. This is particularly the case for ODEs whose solution requires the same order and step size throughout most of the solution domain. As it is explained more clearly in the next chapter, the ODEs that appear in the models of the dynamics of space structures belong to this case. Another serious problem with the selection of variable order and step size is the complexity that is introduced in the programming of the processors. The conclusion is that variable order and step size should be used only in certain unusual cases. This dissertation is limited to the case of using constant order and step size, except under special conditions such as the solution of the ODEs across discontinuities.

It should be realized that the solution of different ODEs may have different computational needs. As a result, different algorithms or different formulas of the same algorithm may be suggested as optimal for each ODE of the system. This would result in the partition of the entire ODE system into groups of ODEs each of which would be solved with different means. Algorithms that are used with different step size and order for different groups of ODEs are called "multi-rate" in distinction to "single-rate" algorithms. Multi-rate implementation of algorithms is discussed in detail in Chapter 3. The use of different order and step sizes can be handled relatively easy. The use of different algorithms however, may be very complicated to implement. With the exception of special cases, the same algorithm should be used for all the groups of ODEs. The approach that is proposed in this chapter is limited to the use of the same algorithm for the entire ODE system. The extension to the case of selecting different algorithms for different ODEs is fairly apparent. As explained in the next section, multiple application of the procedure must be performed for different combinations of candidate algorithms.

2.3 Development of the procedure

According to the previous section, the algorithm selection procedure must:

1. Partition the entire ODE system into groups of ODEs that require the same means for their solution.

2. Suggest the algorithm that should be used for the entire ODE system.

3. Specify the order and step size of the particular formula of the algorithm that should be used for each of the groups of ODEs.

The criteria for the development of the algorithm selection procedure were stated in the introductory chapter as:

1. Maximization of the speed of the solution of the ODE system.

2. Simplicity of the procedure.

A procedure that meets these criteria should select an algorithm which provides a sufficiently accurate solution of the ODE system with the highest speed. In addition, this procedure should be simple and modular enough so that it is easy to automate and should select an algorithm relatively rapidly.

The selection of the most appropriate algorithm including its order and step size for each group of ODEs, can be based on the comparison of the performance of a set of candidate algorithms. The performance of an algorithm can be measured as a combination of its:

1. Speed

2. Accuracy

3. Stability

The definition of this combination depends on the simulation requirements. A simple but sufficient approach is the linear combination. With this approach, the performance P of an algorithm can be expressed as the following weighted sum of its speed S, its accuracy E and its stability Q:

$$P = w_S\, S + w_E\, E + w_Q\, Q \tag{2.1}$$

The weights w_S, w_E and w_Q are the same for every algorithm and should be left as constant parameters of the algorithm selection procedure. The assignment of these weights is left to the user.

The speed S of an algorithm can be measured by the inverse of the time to compute one point of the solution of all the ODEs relative to the real time, averaged over all the points of the solution. The time to compute one point of the solution depends on the order of the formula. The step size at every point of the solution, can be a measure of real time. For an algorithm to be able to provide the solution at real time speed, it has to be able to compute every succeeding point of the solution within a time that does not exceed the corresponding step size. If the time required by the algorithm to obtain the *p*th point of the solution of the *i*th ODE is $\tau_{ip}(m_{ip})$, there are N ODEs and the solution consists of n points, then the speed of the algorithm can be approximated as:

$$S = \frac{1}{n} \sum_{p=1}^{n} \min_{1 \le i \le N} \left\{ \frac{\{h_{ip}\}}{\max\limits_{1 \le i \le N} \{\tau_{ip}(m_{ip})\}} \right\} \tag{2.2}$$

The speed increases with the step size and decreases with the order. The speed of an algorithm can therefore be controlled by tuning its order and step size. This control is essentially equivalent to the selection of the appropriate formula of the algorithm. As explained in the preceding section, only constant order and step size are to be considered, except in special cases. Then the speed of the algorithm can be approximated as:

$$S = \frac{1}{n} \sum_{p=1}^{n} \min_{1 \le i \le N} \left\{ \frac{\{h_{i}\}}{\max\limits_{1 \le i \le N} \{\tau_{ip}(m_{i})\}} \right\} \tag{2.3}$$

The evaluation of the speed of an algorithm has to incorporate the multiprocessor implementation of the algorithm. An important product of the procedure for the multiprocessor implementation of algorithms, presented in Chapter 6, is the evaluation of the speed of the algorithms.

The accuracy E of an algorithm can be measured by the inverse of the error in the solution due to the approximations introduced by the formulas of the algorithm, averaged over all ODEs. Typical measures of this error include:

1. The error l_{ip} introduced at each point *p* of the solution of the *i*th state variable. This is termed the local error of the algorithm. It can always be expressed as:

$$l_i(t_p) = \beta_{ip}\, h_{ip}^{m_{ip}}\, x_i(\xi_p)^{(m_{ip}+1)} \quad \text{where} \quad t_{p-k_i} < \xi_p < t_p, \quad k_i > 0 \tag{2.4}$$

   where the coefficient β and the integer k are characteristics of the formula used by the algorithm and depend on the order of the formula. The local error essentially represents the remaining term of the approximate formula. When the local error is measured as a percentage of the magnitude of the solution $x_i(t_p)$ at the particular point, is termed the relative local error.

2. The accumulated local error over the entire solution. This is termed the global error g of the algorithm. When the global error is measured as a percentage of the magnitude of the solution $x_i(t_p)$ at the particular point, is termed the relative global error G.

The selection of the error measure depends on the simulation requirements, or equivalently on the kind of accuracy required for the solution. The algorithm selection procedure is not sensitive to the particular error measure. Therefore, the adaptation of a particular error measure does not reduce the generality of the procedure. The relative global error is considered here, because it is the most representative. The accuracy can then be expressed as:

$$E = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{\frac{1}{n}\sum_{p=1}^{n} \left| \beta_{ip}\, h_{ip}^{m_{ip}+1} \left[\frac{x_i(\xi_{ip})^{(m_{ip}+1)}}{x_i(t_p)} \right] \right|} \qquad \text{where} \quad t_{p-k_i} < \xi_{ip} < t_p, \quad k_i > 0$$

or for constant order and step size:

$$E = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{\left|\beta_i\, h_i^{m_i+1}\, \Psi_i\right|} \tag{2.5}$$

$$\Psi_i = \frac{1}{n}\sum_{p=1}^{n} \frac{x_i(\xi_{ip})^{(m_i+1)}}{x_i(t_p)}$$

Similar to the speed, it is apparent that it is possible to control the accuracy of an algorithm by tuning its order and step size. This control is equivalent to the selection of the appropriate formula. It should be realized that there exist an upper bound on the accuracy of each algorithm. Therefore, some algorithms may not have any formulation that can provide the desirable accuracy and therefore have to be excluded from further consideration.

The stability Q can be measured by the rate of error accumulation throughout the computation. If an algorithm accumulates error at a high rate, the solution diverges from the actual one and the algorithm is unstable. A stable algorithm on the other hand, even if its accuracy is low, does not accumulate error at a high rate. As a result, it provides a solution that converges to the correct one.

A standard measure of the stability of an algorithm of a particular order and step size is the "stability region". The stability region of an algorithm refers to a linear ODE:

$$\dot{x} = \lambda x \tag{2.6}$$

where λ in general is complex. Any numerical algorithm applied to such ODE can be formulated as:

$$x_{p+1} = \Theta(h\lambda)x_p \tag{2.7}$$

The stability region of an algorithm is the locus of points in the complex plane $s=h\lambda$ that:

$$\left|\Theta(s)\right| \leq 1 \tag{2.8}$$

The boundary of the stability region in the direction of the angle $0 \leq \theta \leq 2\pi$, is defined as the locus of the points $s(\theta)$ such that:

$$\Theta(s) = 1 = e^{j\theta} \tag{2.9}$$

The solution of (2.9) requires the solution of a linear polynomial equation. It should be recognized that λ is the eigenvalue of the ODE (2.6). An algorithm with a specific step size cannot provide a stable enough solution of an ODE whose eigenvalue λ violates (2.8). If such violation occurs, $h\lambda$ lies outside the stability region. If the point $h\lambda$ lies within the stability region, the algorithm has the effect of damping the solution. This prevents the divergence of the solution with the small accuracy cost introduced by the damping. The closer to the boundary of the stability region this point $h\lambda$ is, the lighter the damping. In the case of a linear system of N ODEs with eigenvalues $\{\lambda_j, j=1...,N\}$, all the points $\{h_i\lambda_j, i,j=1,...,N\}$ must fall inside the stability region.

The concept of the stability region can be extended to a nonlinear ODE system by referring to its linearized equivalent. The stability of an algorithm to solve a nonlinear ODE cannot be assured on the basis of the stability region of the equivalent linearized ODE. A trial and error selection of the order and the step size has to be performed. Every trial consists of the actual solution of the nonlinear ODE with a new doubled or halved order and step size. The order and step size that are selected on the basis of the stability region of the linearized ODE, are usually an excellent starting point of the trial and error process. In most cases it is unnecessary to improve this initial order and step size.

The stability of an algorithm depends on its order and step size. Therefore, it is possible to control the stability of an algorithm by tuning its order and step size. Upper limits of the required stability may disqualify some algorithms from further consideration. The stability region of an algorithm is a nonlinear function of its order and a linear function of its step size. Its dependence on the order is mostly monotonous, either increasingly or decreasingly, depending on the algorithm. For different order, it has a different shape. It is always decreasingly monotonous with the step size. For a specific order the stability region is enlarged or diminished proportionally to the step size. The selection of the largest order and the smallest step size of an algorithm that provides a stable solution of a given ODE system is equivalent to the selection of the formula with the smallest stability region that includes all the points $\{h_i\lambda_j,\ i,j=1,\ldots,N\}$. In addition, the stability region should be such that these points are not far from its boundary.

If one or more of the points $\{h_i\lambda_j,\ i,j=1,\ldots,N\}$ does not fall in the stability region of an algorithm, the stability of this algorithm should be counted as zero. Otherwise, the stability of an algorithm can be measured as the amount that its stability region exceeds the absolute minimum stability region required, as defined by the points $\{h_i\lambda_j,\ i,j=1,\ldots,N\}$. Assuming constant order and step size for each state variable x_i, this corresponds to the mathematical expression:

$$Q = \begin{cases} \min\limits_{1 \le i \le N} \{ \sum\limits_{j=1}^{N} s_i(\theta_j) - h_i\lambda_j \} & \text{if for all j: } s_i(\theta_j) \ge h_i\lambda_j \\ 0 & \text{else} \end{cases} \qquad (2.10)$$

where θ_j is the angle direction of the eigenvalue λ_j. This definition of stability reflects the safety margin of the algorithms.

For a specific algorithm, the selection of the order and the step size for a group of ODEs is equivalent to the selection of a particular formula of the algorithm. Recall that for each algorithm, the use of a different number of terms results in a different formula. Therefore, there is an infinite number of formulas for each algorithm. To minimize the number of comparisons, a minimum number of the most competitive formulas has to be selected for each algorithm. This filtering of formulas can be based either on the speed bound, S_u, or on the accuracy and stability bounds, E_u and Q_u respectively. In simulations, the speed criterion is usually the most important. Then candidate algorithms should be compared primarily on the basis of their speed, and the filtering of the formulas should be based on the accuracy and stability bounds. For each algorithm, the smallest set of formulas that provides accuracy and stability that is not below the limits E_u and Q_u, should be selected first. Then the performance of each of the selected formulas should be evaluated using (2.1). Finally, there should be a comparison of the performance of all the formulas that have been selected for all the algorithms.

According to (2.2), the maximization of the speed of an algorithm requires the use of a formula with the maximum step size and the minimum order that the restriction of the accuracy and stability limits permit. Then the filtering of the formulas using the accuracy and stability bounds can be performed in the following way.

For each algorithm, the formulas with the minimum order and the maximum step size for each group of ODEs that satisfy the accuracy and stability limits are selected. In most cases, only one formula will correspond to the optimum pair of step size and order, and therefore pass this filtering test. An algorithm that has no formula that can satisfy the accuracy and stability limits should not be considered further. Similarly, a particular formula of a candidate algorithm that satisfies the accuracy and stability limits, but does not satisfy the speed limit, should be excluded from further consideration.

The partitioning of the ODE system into groups of ODEs requires the determination as to which ODEs require the same means for their solution. As explained in the next section, as well as in the next chapter, the selection of an algorithm and the specification of its order and step size, depends on some of the characteristics of the solution, including the frequency content and the damping of the oscillations of the solution of each ODE. A frequency analysis of the solution can be used to find the groups of ODEs with similar frequency content. An eigenvalue analysis of the linearized equivalent of the ODE system can provide an estimate of the frequency content of the ODE system. An ODE system with eigenvalues $\{\lambda_j = \sigma_j + j\omega_j, j=1,\dots,N\}$ has a solution which contains the frequencies $\{f_j = \frac{\omega_j}{2\pi}, j=1,\dots,N\}$. If the real solution is available, a Fourier analysis of this solution can provide an estimate of the frequency content. Test solutions can be used for this analysis.

2.4 Description of the procedure

Assume that L algorithms have been selected as candidates to solve a given ODE system. According to the approach suggested in the previous section, the algorithm selection procedure can follow the scheme shown in Figure 2.1. The COMPARISON module requires simply the computation of the maximum performance of the L algorithms. The PERFORMANCE EVALUATION module of each algorithm is shown in Figure 2.2.

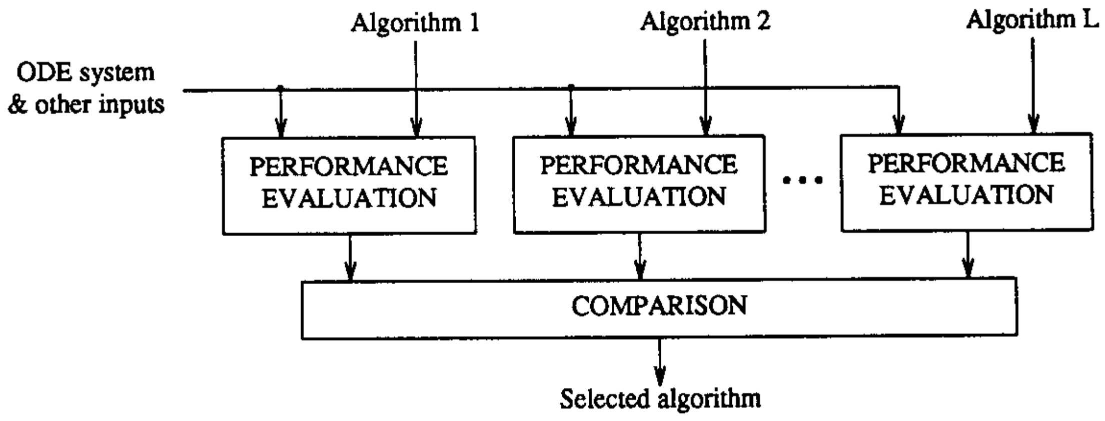


Figure 2.1: The procedure for the selection of an algorithm

Let us discuss each of the three modules of Figure 2.2 in more detail. The module of the SPEED EVALUATION of the algorithm for the optimized orders and step sizes, shown in Figure 2.2, is performed together with the generation of its multiprocessor implementation. This is discussed in Chapter 4. The evaluation of the speed is based on the entire ODE system rather than on an individual group of ODEs. The module of the COMPUTATION OF THE PERFORMANCE can be based on the expression (2.1).

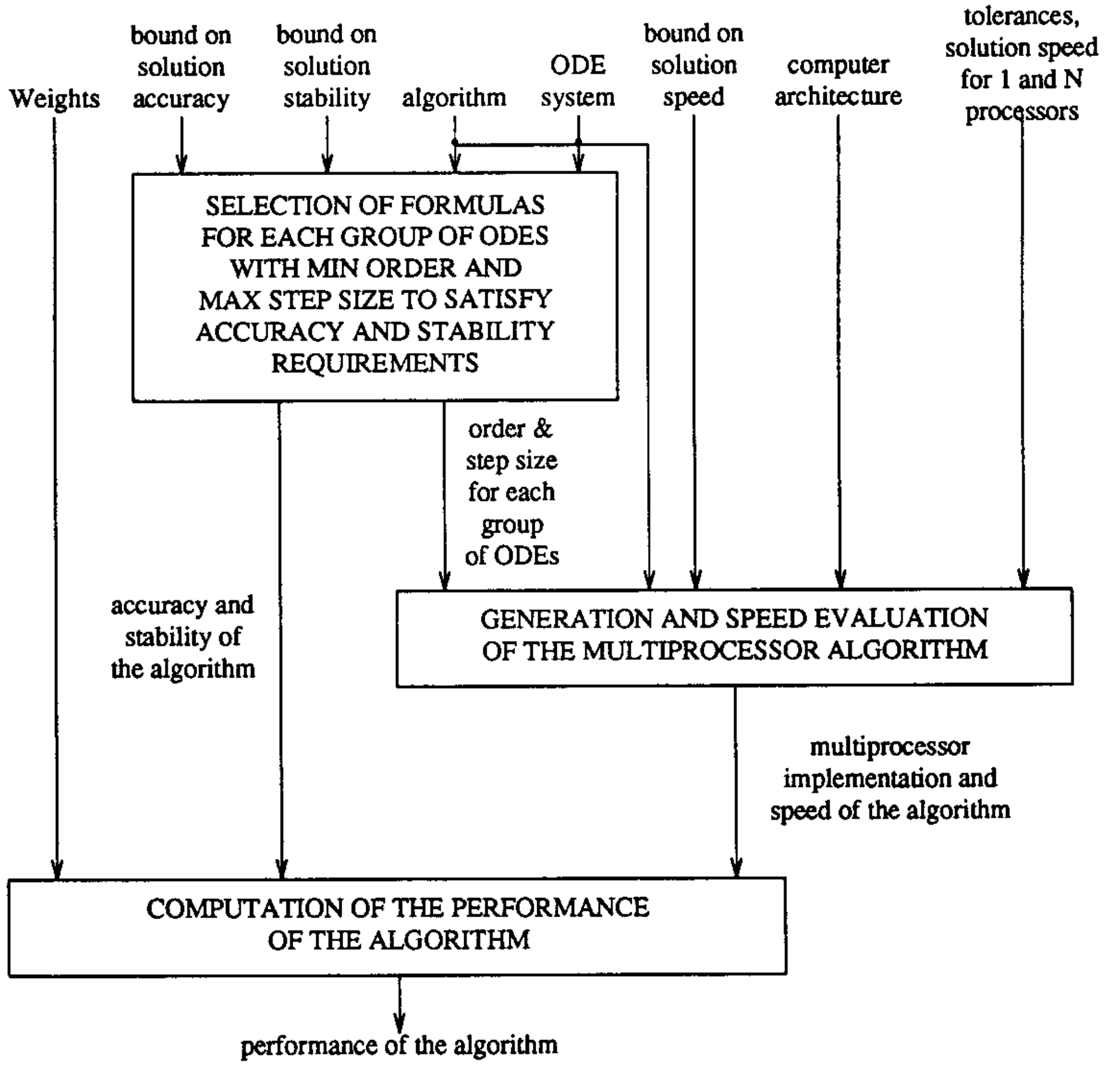

Weights
bound on solution accuracy
bound on solution stability
algorithm
ODE system
bound on solution speed
computer architecture
tolerances, solution speed for 1 and N processors
SELECTION OF FORMULAS FOR EACH GROUP OF ODES WITH MIN ORDER AND MAX STEP SIZE TO SATISFY ACCURACY AND STABILITY REQUIREMENTS
order & step size for each group of ODEs
accuracy and stability of the algorithm
GENERATION AND SPEED EVALUATION OF THE MULTIPROCESSOR ALGORITHM
multiprocessor implementation and speed of the algorithm
COMPUTATION OF THE PERFORMANCE OF THE ALGORITHM
performance of the algorithm

Figure 2.2: Performance evaluation of an algorithm

The module of the SELECTION OF FORMULAS partitions the entire ODE system into, say K, groups of ODEs that require the same means for their solution. Then it selects the formula of the minimum order and the maximum step size for each of the groups of ODEs. Finally, it estimates the accuracy and the stability of the algorithm for the particular orders and step sizes that were selected for the K groups of ODEs. Figure 2.3 shows the implementation of the module.

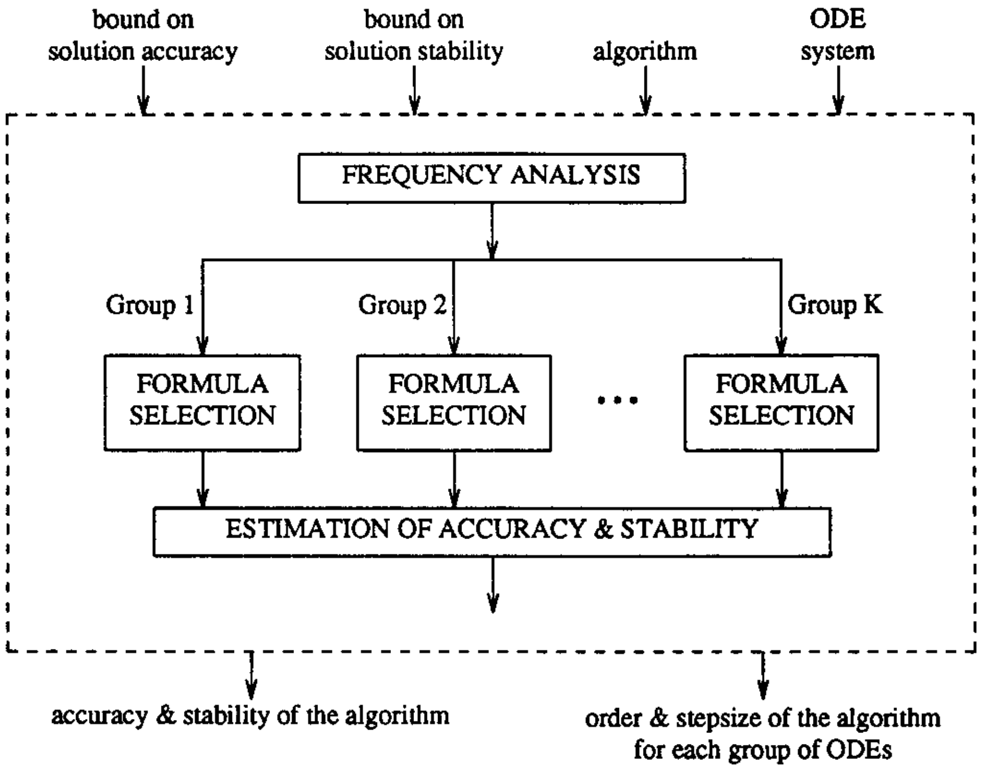


Figure 2.3: Selection of formulas to satisfy accuracy and stability requirements

The FREQUENCY ANALYSIS can be based on the eigenvalue analysis of the linearized equivalent of the ODE system. The ESTIMATION OF ACCURACY & STABILITY can be based on (2.5) and (2.10). For each group of ODEs, the FORMULA SELECTION or equivalently the selection of the order and step size can be be formulated as the following nonlinear optimization problem:

$$\text{Find a set } \{(m_i,h_i),\ i=1,\dots,N\} \text{ that maximizes } S \tag{2.11}$$

under the constrains: $E \geq E_u$ and $Q \geq Q_u$

where S, E and Q are defined in (2.3), (2.5) and (2.10). This optimization problem can be solved heuristically by decreasing m_i and increasing h_i until a maximum of S is realized. The monotonicity of the dependence of the accuracy and the stability on the order and the step size guarantees the uniqueness of the optimal pair or order and step size. The step size affects the speed more than the order. Halving the order reduces the amount of computation needed for the computation of each point of the solution, while doubling the step size eliminates 50% of the points of the solution that need to be computed. Therefore, the maximization of the step size should be given priority over the minimization of the order. This means that within the trial and error process of the determination of the optimal step size and order, for each order there should be an exhaustive search for the longest step size before the order is minimized.

The initial m_i should be the minimum order that the particular algorithm permits. Most of the algorithms have formulas of first order. The initial h_i should be the maximum step size that the expected frequency content of the solution permits. It was explained earlier that a rough but simple and fast estimate of the frequency content of the solution can be obtained from the imaginary part of the eigenvalues of the linearized ODE system. The maximum frequency, say f_{max}, that appears in the solution, dictates the initial step size. The accurate representation of the solution requires that at least 10 points over the period of the oscillation with the highest frequency be obtained. Then:

$$h_{i_{max}} = \frac{1}{10 f_{i_{max}}} \tag{2.12}$$

An eigenvalue analysis is therefore needed at the beginning of each FORMULA

SELECTION module to estimate the maximum possible step size. Any of the commercially available software, such as the EISPACK, can be used to compute the eigenvalues of the specific algorithm. This can be part of the FREQUENCY ANALYSIS that has to be performed for the partition of the ODE system into groups of ODEs. The entire procedure for the FORMULA SELECTION is shown in Figure 2.4.

To determine the minimum order, a table can be maintained that indicates the minimum order that each of the candidate algorithms permits. The computation of the maximum step size can be based on (2.12). The estimations of the accuracy and the stability can be based on (2.5) and (2.10). The decisions of acceptability of the accuracy and the stability can be based on the satisfaction of the constrains in (2.11). Different type of ODE systems may require different approaches for the computation of E and Q. An approach for ODE systems that appear in space structures is presented in the next chapter. Finally, the decision "WORSEN STEPSIZE ?" is based on the comparison of the longest step size that the present and the previous values of the order permit.

2.5 Validation of the procedure

The completion of the presentation of the procedure requires a validation of the procedure with reference to the criteria of Section 2.3. In Chapter 1, it was pointed out that the selection of the appropriate algorithm contributes to the maximization of the solution speed by minimizing the computational load. The other mean of maximizing of the solution speed, that is the optimization of the distribution of this load, is to be achieved by the multiprocessor implementation of the algorithm that is selected. The use of a formula of minimum order and maximum step size is

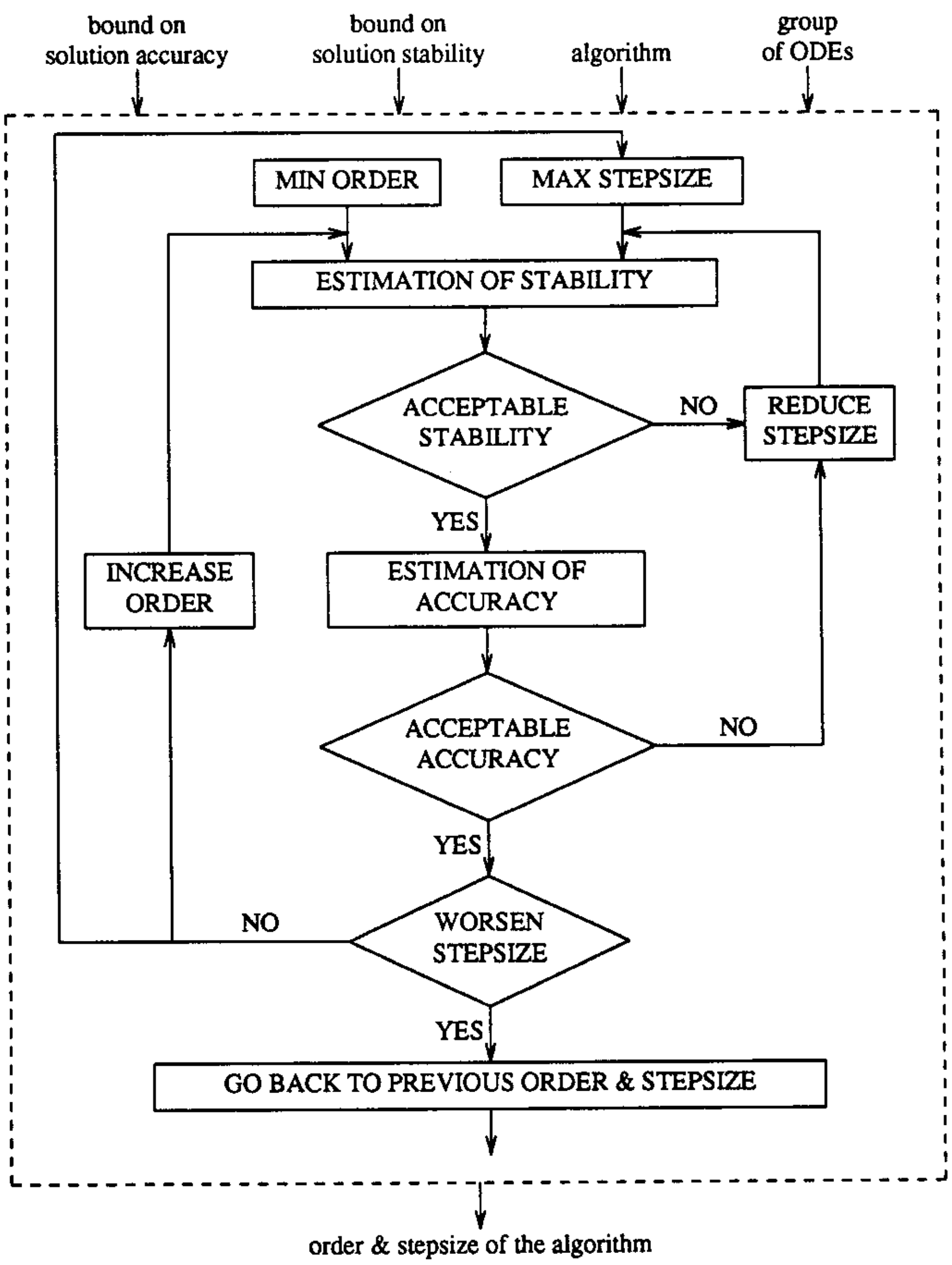


Figure 2.4: Selection of the formula of an algorithm for a group of ODEs

the most effective mean to minimize the computational load. Therefore, the procedure that is proposed indeed selects the algorithm that provides the highest solution speed. In addition, the procedure is simple, quantitative, systematic and modular. The modules that correspond to the estimate of the accuracy and the stability of the algorithms are the only ones that may introduce some high complexity. This complexity can be controlled however, by the approach that is taken for these estimations. Finally, the speed with which the procedure selects an algorithm is also controllable, because it depends also on the complexity of the approach that is taken for the accuracy and stability estimations.

2.6 Summary

This chapter is devoted to the high level presentation of the algorithm selection procedure, applicable to any type of ODE system. The procedure partitions the entire ODE system into groups of ODEs that require the same means for their solution. It then selects the appropriate algorithm to solve the entire ODE system, and specifies the order and the step size of the formulas that should be used for each of the groups of ODEs. The selection is based on the comparison of the performance of the candidate algorithms, as shown in Figure 2.1. The evaluation of the performance of each algorithm is shown in Figure 2.2. The performance evaluation of each algorithm includes the selection of the optimal formula that provides a sufficiently accurate and stable solution with the highest speed, as shown in Figure 2.3. This is the formula with the optimum combination of order and step size. For each of the groups of ODEs, the selection of the optimum formula for a particular algorithm is shown in Figure 2.4. The procedure is simple, systematic, quantitative and sufficiently modular so as to be easy to automate. Therefore, the procedure meets the criteria of maximization of the speed of the solution of the ODE system and simplicity.

3. ALGORITHMS FOR SIMULATION OF SPACE STRUCTURES

The preceding chapter introduced in a high level a procedure that selects algorithms to solve ODE systems. This chapter is devoted to the specialization of the procedure for ODE systems that appear in the models of the dynamics of space structures. It offers suggestions for the implementation of the high level modules for the specific ODE systems. The presentation of the procedure in this chapter can be considered also as an illustrative demonstration for a specific but representative case. The first section of this chapter includes an analysis of the computational characteristics of the ODE systems that appear in the models of space structures. This is followed by a discussion on the requirements for the solution of these ODE systems. These discussions lead, in the next section, to suggestions for the implementation of the procedure for the performance evaluation of the algorithms. The following section is devoted to the selection of the set of candidate algorithms from a large assortment of widely used algorithms. The next section proceeds to a more detailed comparison of the candidate algorithms. In particular it discusses the performance of the algorithms in computing the solution across discontinuities. The final section explains the multirate implementation of the algorithms.

3.1 Characteristics of the ODE systems

The evaluation of an algorithm to solve an ODE system requires an analysis and a measurement of the computational requirements for the solution of the ODE system. These requirements are imposed by the following characteristics of the ODE system:

1. Size and complexity.

2. Frequency content and damping of the oscillations of the state variables.

The systems that appear in the models of the dynamics of space structures usually consist of few hundreds of ODEs. In [Gluc85], for example, it is reported that a simplified modeling of the space shuttle of Figure 1.1 resulted in 130 ODEs. These large ODE systems are also highly complex. The main source of their complexity are several type of nonlinearities, including:

1. Simple nonlinear mathematical expressions of the state variables, such as squares or higher powers. These nonlinearities may appear in the elements of both the matrix A and the vector b.

2. Trigonometric expressions of the state variables or of time, due to the oscillating nature of the dynamics of the space structures and due to coordinate transformations. These nonlinearities may also appear in the elements of both the matrix A and the vector b.

3. Functions of the state variables or of time, that contain discontinuities, such as steps, absolute values, pulses, hystereses, dead zone delimiters and relays. Such nonlinearities may be introduced by:

    a. The sudden application or removal of the excitations. These appear in the elements of the vector b only.

    b. The property of the materials. These may appear in the elements of both the matrix A and the vector b.

    c. The control systems. These may also appear in the elements of both the matrix A and the vector b.

The evaluation of these nonlinear expressions is usually very time consuming. In many simulations it is the major part of the entire computation. Several approximate techniques can be used to speedup the evaluation of some of them, including table lookups.

The frequency content of the oscillation of the state variables is usually wide. In addition, the damping is usually very slow. In a typical case, such as the space shuttle of Figure 1.1, the oscillations of bodies may contain frequencies in the range 0.5 - 100 Hz, and may be damped by 99% after 300 secs.

The conclusion is that the ODE systems that appear in the mathematical models of the dynamics of space structures are:

1. Large
2. Highly nonlinear
3. Highly oscillating
4. Lightly damped
5. Formulated as $A(x,t)\dot{x}=b(x,u,t)$

3.2 Requirements for the solution of the ODE systems

As stated in Chapter 1, the transient solution of the ODE systems with a high speed and high accuracy has to be obtained. Let us analyze these requirements.

Speed

Often, a real time solution is required. Each point of the solution then has to be computed in time that is at most equal to the corresponding step size. The size and the complexity of the ODE system together with the small size of the step size that has to be used to maintain high accuracy, as discussed below, may require a large number of processors to achieve real time computation. Such a large number of processors may not be available however, or their power may be wasted during most of the computation because many of the processors may stay idle. Then it may be preferable for a user to relax the real time requirement in favor of the cost.

Accuracy

Reasonable accuracy is always required. Often the global relative error must be limited to 1%, so that the needed characteristics of the oscillations of the state variables can be determined with the required accuracy.

The requirement for high accuracy has the general impact that for each state variable all the important frequency components should appear accurately in the solution. Components with relatively small amplitude can be ignored because of their small contribution to the solution. In addition, the step size at each point of the solution should be small enough to follow the highest frequency component F that is to appear accurately. If it is assumed that at least 10 points per period are required to represent with sufficient accuracy one period of an one frequency oscillation, the maximum step size that can be used for the *i*th ODE is:

$$h_{i_{max}} = \frac{1}{10F_i} \tag{3.1}$$

However, the step size is bounded more strictly by (2.12). This is because the highest frequency f_{max} that appears in the solution is larger than or equal to the highest frequency F that must be represented accurately in the solution. An algorithm with the appropriate characteristics concerning stability should damp the components with frequencies larger than F fast enough so that their presence in the ODE system can be ignored. Then (3.1) can be used instead of (2.12) in the estimation of the MAX STEPSIZE in Figure 2.4. The upper bound on the step size expressed by (3.1) implies also the potential use of a different step size for different groups of ODEs having different F. Then a multirate implementation of the algorithms must be used.

The accuracy of the computation for each point of the solution can be controlled relatively easily in most cases. The accumulation of error however, is more difficult to control because of the large number of points of the solution that usually need to be computed. For example, if a 100 Hz frequency has to appear accurately in the solution, the step size cannot be larger than 0.001 secs, according to (3.1). If the physical damping of the oscillations takes 300 secs, then 300000 solution points have to be computed. The stability of such long solution may be poor unless the proper techniques are used.

3.3 Performance evaluation of algorithms

This section is devoted to the presentation of an implementation of some interesting parts of the algorithm selection procedure that is presented in a high level in Chapter 2, the specific case of the dynamic simulation of space structures. In particular, this presentation is concerned with the implementation of the two modules in Figure 2.4, that is the estimation of the accuracy and the stability of the algorithms, and with the definition of the speed, accuracy and stability bounds. The rest of the modules of the algorithm selection procedure, as shown in Figures 2.1, 2.2, 2.3 and 2.4, require no further explanation.

Estimation of Accuracy

The accuracy of an algorithm for chosen orders and step sizes can be determined from (2.5). The high derivatives $x_i^{(m_i+1)}$ are the only terms that need further discussion. Their estimation for highly oscillating problems is a crucial step in computing E. The efficient but also accurate estimation of this derivative is however difficult in general. It is still an open problem in numerical analysis. An estimation using "brute force" is the most practical way to provide the derivative fairly accurately. Such an estimation is based on a finite difference approximation of the derivatives, such as:

$$x_i(\xi_{ip})^{(\mu+1)} \approx \frac{x_i(\xi_{ip})^{(\mu)} - x_i(t_{p-k_i})^{(\mu)}}{\xi_{ip}-t_{p-k_i}} \tag{3.2}$$

This requires an apriori test solution and a recursive application of (3.2) for $\mu=1,\ldots,m_i$. Some algorithms incorporate the computation of all or some of the lower derivatives, up to the *m*th derivative. In those cases, (3.2) has to be applied only for $\mu=m_i$. The "brute force" approach is fairly accurate but expensive. A rough but significantly less expensive estimate can be obtained by assuming that the solution $x_i(t)$ of the *i*th state variable of the ODE system $A(x,t)\dot{x}=b(x,u,t)$ can be approximated by the solution $\overline{x}_i(t)$ of the *i*th state variable of the linearized equivalent ODE system. In that case $x_i(t)$ can be approximated by a combination of N decaying oscillations as:

$$x_i(t) \approx \sum_{k=1}^{N} y_{ik}(t) = \sum_{k=1}^{N} \rho_{ik} e^{\lambda_k t} \tag{3.3}$$

$$\rho_{ik} = T_{ik} \sum_{v=1}^{N} T_{kv}^{-1} x_v(0)$$

where the generally complex number ρ_{ik} is a measure of the contribution of the $\lambda_k = \sigma_k + j2\pi f_k$ eigenvalue to the solution of the *i*th state variable. T is the generally complex matrix of the eigenvectors of the system. The eigenvalues and eigenvectors of the ODE system can be obtained from the FREQUENCY ANALYSIS module in Figure 2.3. It can then be shown that:

$$x_i(t)^{(m_i+1)} \approx \sum_{k=1}^{N} w_{ik}(t) y_{ik}(t) \tag{3.4}$$

$$w_{ik}(t) = (\sigma_k^2 + (2\pi f_k)^2)^{\frac{m_i+1}{2}} e^{j(m_i+1)\tan^{-1}\left[\frac{2\pi f_k}{\sigma_k}\right]}$$

Then the estimation of the accuracy can follow (2.5) with:

$$\Psi_i = \frac{1}{n} \sum_{p=1}^{n} \frac{\sum_{k=1}^{N} w_{ik}(\xi_{ip}) y_{ik}(\xi_{ip})}{\sum_{k=1}^{N} y_{ik}(t_p)} \quad \text{where} \quad \xi_{ip} = \frac{t_p + t_{p-k_i}}{2} \tag{3.5}$$

A more simple approximation can be derived from (3.4) by observing that all σs are very small. For oscillations that are damped by 99% after 300 secs, σs would be approximately 0.015. As a result, terms that contain powers of σs are negligible. Furthermore, one can approximate Ψ with its upper bound, which is of the order of $(2\pi f_{max})^{m_i+1}$. In this case however, it should be expected that the estimation of the accuracy, and therefore of the orders and the step sizes, might be a little conservative.

Estimation of stability

The estimation of the stability of an algorithm can be based on (2.10). This includes the computation of the eigenvalues $\{\lambda_j, j=1,...N\}$ which can be performed in the FREQUENCY ANALYSIS module of Figure 2.3. The set of points $\{h_i\lambda_j, j=1,...,N\}$ define the absolutely minimum desirable stability region. Let us determine this minimum stability region for the specific ODE systems of interest.

The eigenvalues of a lightly damped ODE system are nearly imaginary, i.e. they lie close to the imaginary axis on the complex plane. If the eigenvalue with the largest imaginary part is $\sigma+j2\pi f_{max}$, the stability region has to include the area close to the imaginary axis up to $j2\pi f_{max} h_i$. In addition, recall that the algorithm to be used has to provide a solution that represents accurately only the frequency components that make a significant contribution to the solution. This means that if the highest important frequency for the *i*th state variable is F_i, the algorithm must dampen fast enough the components with frequency higher than F_i and dampen as slow as possible the components with frequency lower or equal to F_i. The points up to $j2\pi F_i h_i$ have to be close to the boundary of the stability region, while the points further than $j2\pi F_i h_i$ should lie deeply in the stability region. Figure 3.1 shows the stability behavior that is required. The STABLE areas define the stability region that is needed.

Speed, accuracy and stability bounds

If real time stable solution is required, with relative global error limited to 1%, then from (2.3), (2.5) and (2.10):

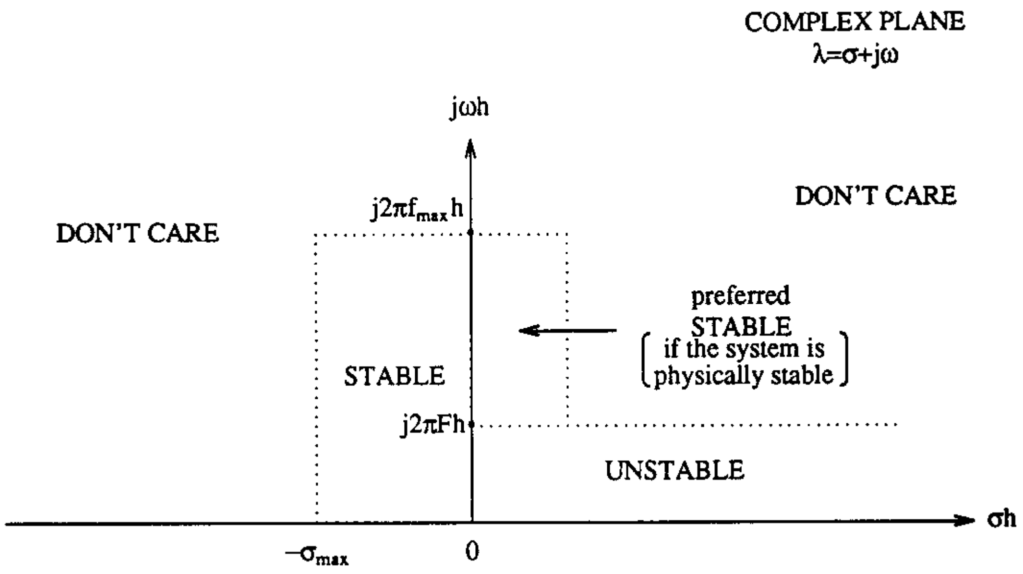


Figure 3.1: The stability region that is required

$$S_u \geq 1, \quad E_u \geq 100 \quad \text{and} \quad Q_u \geq 0 \tag{3.6}$$

3.4 Selection of the candidate algorithms

The most widely used algorithms belong to the following families: [Gear71]

1. Explicit Power Series Expansion

2. Explicit Runge Kutta

3. Explicit Multi-step

4. Implicit Power Series Expansion

5. Implicit Runge Kutta

6. Implicit Multi-step

7. Implicit Multi-step Multi-derivative

A rough qualitative estimate of the performance of all these algorithms is needed to select the candidate algorithms to be compared. The comparison of the speed of the algorithms can be based on the number of inversions of the matrix A(x,t), the number of derivatives of the state vector and the number of iterations they require per step. The comparison of the accuracy of the algorithms can be based on their parameter β. According to (2.5), the smaller this parameter is, the higher the accuracy. Figure 3.10 compares the beta parameters of the algorithms. The comparison of the stability of the algorithms can be based on their stability region. The closer the stability region is to the one shown in Figure 3.1 and the larger it is, the higher the stability. Figure 3.11 compares the stability region of the algorithms close to the imaginary axis.

Single-step algorithms, such as the Power Series Expansion and the Runge Kutta, compute the next point of the solution based on the points only one step back, that is on the previous point. They are formulated as:

$$x_{p+1} = \Phi(x_p) \quad \text{or} \quad x_{p+1} = \Phi(x_{p+1}, x_p) \tag{3.7}$$

Multi-step algorithms compute the next step of the solution based on the points more than one step back, that is on several previous points. They are formulated as:

$$x_{p+1} = \Phi(x_p, x_{p-1}, \ldots, x_{p-k}) \quad \text{or} \quad x_{p+1} = \Phi(x_{p+1}, x_p, x_{p-1}, \ldots, x_{p-k}) \tag{3.8}$$

Multi-step algorithms have more simple formulations. Therefore they usually are faster per step. Single-step algorithms usually allow larger step size however, due to their higher accuracy and stability. Therefore, the overall speed of the single-step algorithms, as defined in (2.2), can be competitive to that of the multi-step ones.

Explicit algorithms are based on formulations that express the next point of the solution explicitly, such as:

$$x_{p+1} = \Phi(x_p) \quad \text{or} \quad x_{p+1} = \Phi(x_p, x_{p-1}, \ldots, x_{p-k}) \tag{3.9}$$

while implicit are based on formulations that express the next point of the solution implicitly, such as:

$$x_{p+1} = \Phi(x_{p+1}, x_p) \quad \text{or} \quad x_{p+1} = \Phi(x_{p+1}, x_p, x_{p-1}, \ldots, x_{p-k}) \tag{3.10}$$

Implicit algorithms are slower per step because they require iterative computation of x_{p+1}, but they usually allow larger step sizes due to their greater stability and accuracy. Therefore their overall speed is competitive with that of the explicit ones.

The rest of this section compares the algorithms based on a qualitative analysis of their speed, accuracy and stability. The conclusion of this comparison is shown in Table 3.1 at the end of the section.

Explicit Power Series Expansion

A standard technique to compute the next point of the solution from the present one is the Taylor series expansion of the solution at the next point [Gear71] as follows:

$$x_{i_{p+1}} = \sum_{\nu=0}^{m_i} \frac{h_i^\nu}{\nu!} x_{i_p}{}^{(\nu)} \tag{3.11}$$

The explicit power series expansion algorithm (EPSE) is an efficient technique to compute the terms of the Taylor series. Although this technique was introduced many decades ago for numerical differentiation, its use in solving ODEs only recently gained popularity, as a result of the contributions of Dr. Halin [Hali83]. To compute the derivatives that appear in the series expansion, semianalytical differentiation of x_i is employed, rather than the classical analytical methods which may be difficult and time consuming. This semianalytical differentiation is based on the decomposition of the mathematical expression of $\dot{x}_i$ into a number of elementary expressions whose derivation is easily performed recursively. As a simple example consider the ODE:

$$\dot{x} = xe^x + \sin x \tag{3.12}$$

$\dot{x}$ can be decomposed into p+r, where p=xq, q=e^x and r=sinx. Then the derivatives of x can be computed recursively in the following sequence:

$$\begin{aligned}
q^{(v)} &= \sum_{j=0}^{v-1} \binom{v-1}{j} x^{(v-j)} x^{(j)}, \quad v\geq 1 \\
p^{(v)} &= \sum_{j=0}^{v} \binom{v}{j} x^{(v-j)} q^{(j)}, \quad v\geq 0 \\
r^{(v)} &= \sum_{j=0}^{v-1} \binom{v-1}{j} x^{(v-j)} s^{(j)}, \quad v\geq 1 \\
s^{(v)} &= -\sum_{j=0}^{v-1} \binom{v-1}{j} x^{(v-j)} r^{(j)}, \quad v\geq 1 \\
x^{(v)} &= p^{(v)} + r^{(v)}, \quad v\geq 0
\end{aligned} \tag{3.13}$$

For ODE systems of the form $A\dot{x}=b$, two levels of recursion have to be applied. In the upper level the following recursion has to be introduced for the computation of the derivatives of the vector x:

$$x_p^{(v)} = A_p^{-1} z_{vp} \quad \text{where} \quad z_{vp} = \begin{cases} b_p, & v=1 \\ \left[b_p^{(v-1)} - \sum_{j=1}^{v-1} \binom{v-1}{j} A_p^{(j)} x_p^{(v-j)} \right], & v>1 \end{cases} \tag{3.14}$$

It is important to notice that the vector z can be computed recursively as:

$$z_{v_p} = z_{v-1_p}{}' - A_p' x_p^{(v-1)} \tag{3.15}$$

In the lower level, recursion has to be introduced in the computation of the derivatives of the elements of the matrix A and the vector b. In addition, a more efficient evaluation of the series (3.11) can be based on the nested computation of polynomials as follows:

$$x_{i_{p+1}} = y_0 \quad \text{where} \quad y_v = \begin{cases} x_{i_p}{}^{(v)} + \dfrac{h_i}{v+1} y_{v+1}, & v \le m_i \\ 0, & v > m_i \end{cases} \tag{3.16}$$

Each point of the solution of x_i requires one inversion of the matrix A shared by all state variables, the recursive computation of the derivatives of the elements of the *i*th row of the matrix A and the *i*th element of the vector b, the computation of the derivatives of x_i based on (3.14), and the evaluation of the new x_i based on (3.16). For most large ODE systems the inversion of A is the most time consuming part of the computation of the next point of the solution. The need for inversion of A only once is a great advantage of the EPSE algorithm comparing to most other algorithms which require the inversion of A many times for each point of the solution.

The parameter β of the EPSE algorithm, shown in Figure 3.10 in comparison with the other algorithms, is:

$$\beta_i = \frac{1}{(m_i+1)!} \tag{3.17}$$

Figure 3.10 indicates that the accuracy of the EPSE algorithm is among the highest. The stability region of the EPSE algorithm is shown in Figure 3.2, and compared with other algorithms in Figure 3.11. Comparing Figure 3.2 to Figure 3.1, it is recognized that the stability region of the EPSE algorithm has the desirable shape and that the minimum order has to be 3. Figure 3.11 indicates that the stability of the EPSE algorithm is among the highest. Both the accuracy and stability increase monotonically with order and decrease monotonically with step size. This is another important advantage of the EPSE algorithm, because the selection of the order and the step size to satisfy the accuracy requirements does not contradict the selection criterion to satisfy stability requirements. Therefore, the optimal order and step size are closer to the desirable values, that is the absolute minimum and maximum respectively.

The complexity of the algorithm to compute one point of the solution based on (3.14) and (3.16) is relatively low. The recursive computation of the derivatives, the need to invert A only once, and the relatively small order contribute significantly to the lowering of this complexity. In addition, the relatively high accuracy and stability of the algorithm allows the use of fairly large step sizes. As a result, the speed of the algorithm, according to (2.2) is in general fairly high. This leads to the conclusion that the EPSE algorithm definitely qualifies as a candidate for comparison.

Explicit Runge-Kutta

The Explicit Runge Kutta (ERK) algorithm replaces the computation of the derivatives in the Taylor series (3.11) with an averaging of values of the first derivative at m_i points between the present and the next step, x_p and x_{p+1} respectively

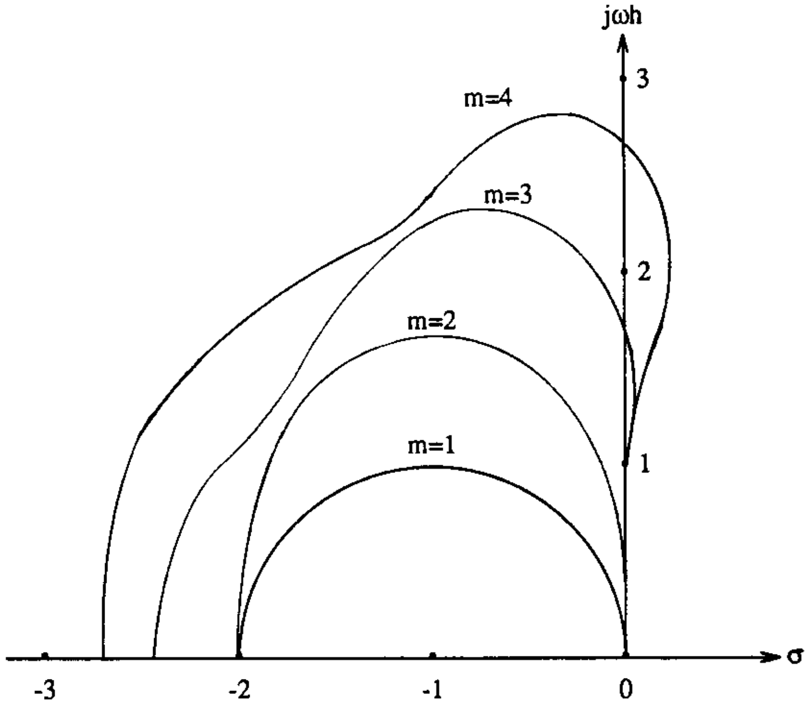


Figure 3.2: Stability of the Explicit Power Series Expansion algorithm
The algorithm is stable inside the curves

[Gear71], as:

$$x_{i_{p+1}} = x_{i_p} + \sum_{\nu=0}^{m_i-1} \gamma_\nu \kappa_{i\nu} \quad \text{where} \quad \kappa_{i\nu} = \begin{cases} h_i \dot{x}_i(t_p), & \nu=0 \\ h_i \dot{x}_i(t_p + \sum_{\mu=1}^{\nu} \zeta_{\nu\mu} \kappa_{\mu-1}), & \nu \geq 1 \end{cases} \qquad (3.18)$$

The κ and γ parameters are selected so that (3.18) matches the Taylor series (3.11). Thus the ERK algorithm has accuracy and stability performance similar to the EPSE. The computation of (3.18) however, is significantly more time consuming than the combination of (3.14) and (3.16), because it requires m_i inversions of A, one for each of the κs. The inferiority of the ERK algorithm when compared to the EPSE is apparent. Therefore, the ERK is not a candidate algorithm.

Explicit Multi-step

These algorithms use polynomial interpolation formulas to compute the next point of the solution from a number of previous points. The number of previous points defines the order of the formula. The most widely used explicit multi-step algorithm is the following Adams-Bashforth (AB) formula:

$$x_{i_{p+1}} = x_{i_p} + h_i \sum_{j=p}^{p-m_i+1} \gamma_j \dot{x}_{i_j} \tag{3.19}$$

Each point of the solution of x_i requires one inversion of the matrix A shared by all state variables, and the evaluation of the new x_i based on (3.19).

The parameter β is shown in Figure 3.10. The stability region is shown in Figure 3.3. Figures 3.10 and 3.11 indicate that both the accuracy and the stability of the AB algorithm for the same order and step size are poor compared to the EPSE. The need of a high order to control the accuracy has to compete with the need for a smaller order to control the stability. Only a significantly smaller step size can satisfy the accuracy and the stability requirements. This is a serious disadvantage in terms of speed. However, the time to compute one point of the solution is smaller than the EPSE. This can keep the AB algorithm in a competitive level, and therefore qualify it to be a candidate algorithm.

Implicit Power Series Expansion

The Implicit Power Series Expansion (IPSE) algorithm extends the EPSE by including a series expansion of x_{p+1} as well. Instead of the Taylor series expansion, uses an expansion based on the Pade approximation of e^{-z}. It is formulated as:

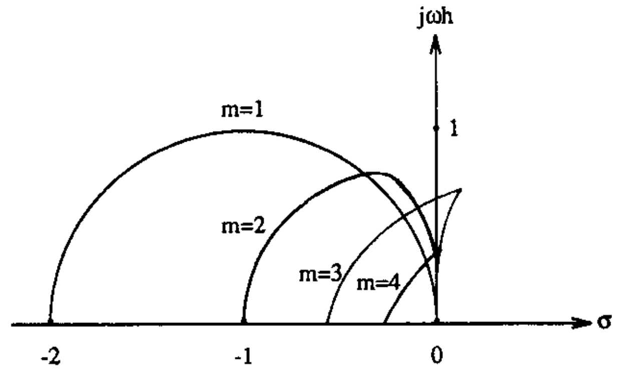


Figure 3.3: The stability region of the Adams Bashforth algorithm
The algorithm is stable within the curves

$$x_{i_{p+1}} = x_{i_p} + \sum_{v_1=1}^{m_{1_i}} d_{v_1} h_i^{v_1} x_{i_p}{}^{(v_1)} + \sum_{v_2=1}^{m_{2_i}} (-1)^{v_2+1} e_{v_2} h_i^{v_2} x_{i_{p+1}}{}^{(v_2)} \qquad (3.20)$$

where the d and e coefficients are the coefficients of the nominator and the denominator of the Pade approximation of e^{-z} [Twiz81]. The order of this formula is $m_i=m_{1_i}+m_{2_i}$. Provided that an infinite number of iterations has been applied to obtain an exact solution of (3.20), the β parameter, shown in Figure 3.10, is smaller than $(m_{1_i}+m_{2_i}+1)^{-1}$ resulting in higher accuracy than the EPSE. For a small number of iterations the accuracy of the IPSE algorithm is close to that of the EPSE. The IPSE algorithm is popular because of its strong stability. Figure 3.4 shows the stability region of several IPSE formulas, provided that an infinite number of iterations have been employed to obtain the exact solution of (3.20). The stability region for finite number of iterations is a subset of those shown in Figure 3.4. Formulas with $m_1>m_2$ are always unstable. Formulas with $m_1<m_2$ are too stable to be used because according to Figure 3.1 they would damp the components with frequencies lower than F. Only the formulas with $m_{1_i}=m_{2_i}$ have stability regions that are acceptable,

despite the fact that they do not damp the high frequency components. The high stability of the IPSE algorithms allows large step sizes.

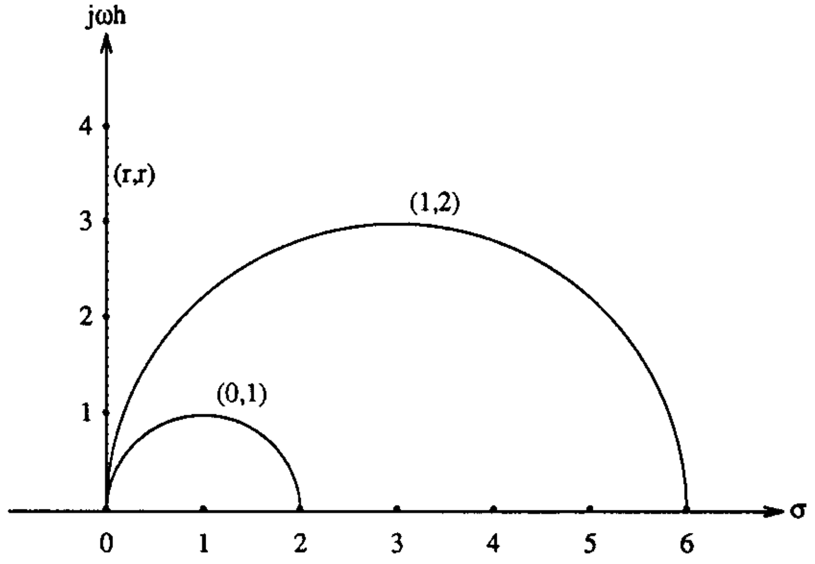


Figure 3.4: Stability of the Implicit Power Series Expansion algorithm
The algorithm is stable outside the curves

The computation of x_{p+1} requires the solution of (3.20). The least time consuming solution can be obtained by the following fixed point iteration:

$$A_p^{(\mu)} x_p^{(\nu_1)^{(\mu)}} = b_p^{(\nu_1-1)^{(\mu)}} - \sum_{j=1}^{\nu_1-1} \begin{bmatrix} \nu_1-1 \\ j \end{bmatrix} A_p^{(j)^{(\mu)}} x_p^{(\nu_1-j)^{(\mu)}}, \quad \nu_1=1,\ldots,m_1 \tag{3.21}$$

$$A_{p+1}^{(\mu)} x_{p+1}^{(\nu_2)^{(\mu)}} = b_{p+1}^{(\nu_2-1)^{(\mu)}} - \sum_{j=1}^{\nu_2-1} \begin{bmatrix} \nu_2-1 \\ j \end{bmatrix} A_{p+1}^{(j)^{(\mu)}} x_{p+1}^{(\nu_2-j)^{(\mu)}}, \quad \nu_2=1,\ldots,m_2$$

$$x_{i_{p+1}}{}^{(\mu+1)} = x_{i_p} + \sum_{\nu_1=1}^{m_{1_i}} d_{\nu_1} h_i^{\nu_1} x_{i_p}{}^{(\nu_1)} + \sum_{\nu_2=1}^{m_{2_i}} (-1)^{\nu_2+1} e_{\nu_2} h_i^{\nu_2} x_{i_p}{}^{(\nu_2)^{(\mu)}} = g_\mu$$

The EPSE algorithm can be used to obtain the initial guess for x_{p+1}. The convergence of fixed point iteration requires that:

$$\left| \frac{dg}{dx_{p+1}} \right| = \left| \sum_{v_2=1}^{m_{2_i}} (-1)^{v_2+1} e_{v_2} h_i^{v_2} \frac{dx_{i_p}^{(v_2)}}{dx_{p+1}} \right| < 1 \qquad (3.22)$$

The satisfaction of (3.22) may impose a very small step size, negating therefore, the stability advantages of the algorithm. The stability region of the algorithm is the subset of the one shown in Figure 3.4 that satisfies (3.22). Figure 3.5 shows the stability region of the (1,1) formula. In case a finite number of iterations is taken, the stability region is a subset of the one shown in Figure 3.5.

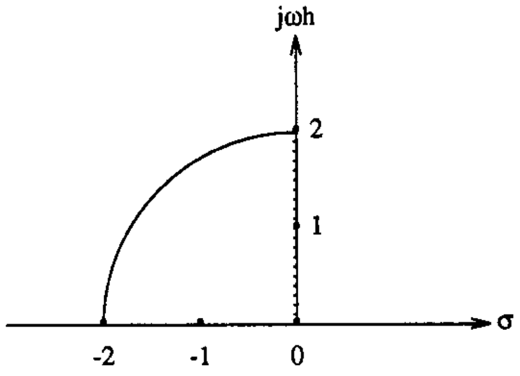


Figure 3.5: Stability of the (1,1) Implicit Power Series Expansion algorithm for fixed point iterations. The algorithm is stable inside the curve

To avoid this limitation, one can use the following Newton iteration:

$$A_p^{(\mu)} x_p^{(v_1)(\mu)} = b_p^{(v_1-1)(\mu)} - \sum_{j=1}^{v_1-1} \begin{bmatrix} v_1-1 \\ j \end{bmatrix} A_p^{(j)(\mu)} x_p^{(v_1-j)(v)}, \quad v_1=1,\dots,m_1 \qquad (3.23)$$

$$A_{p+1}^{(\mu)} x_{p+1}^{(\nu_2)(\mu)} = b_{p+1}^{(\nu_2-1)(\mu)} - \sum_{j=1}^{\nu_2-1} \begin{bmatrix} \nu_2-1 \\ j \end{bmatrix} A_{p+1}^{(j)(\mu)} x_{p+1}^{(\nu_2-j)(\mu)}, \qquad \nu_2=1,\dots,m_2$$

$$\Psi_{\nu_2}^{(\mu)} = \text{N}\times\text{Nmatrix with columns } \frac{\partial A_{p+1}^{(\mu)}}{\partial x_{i_{p+1}}} x_{p+1}^{(\nu_2)(\mu)}$$

$$\Omega_j^{(\mu)} = \text{N}\times\text{Nmatrix with columns } \frac{\partial A_{p+1}^{(j)(\mu)}}{\partial x_{i_{p+1}}} x_{p+1}^{(\nu_2-j)(\mu)}$$

$$J_j^{(\mu)} = \text{N}\times\text{Nmatrix with columns } \frac{\partial x_{p+1}^{(j)}{}^{(\mu)}}{\partial x_{i_{p+1}}} \qquad \text{(Jacobian)}$$

$$B_j^{(\mu)} = \text{N}\times\text{Nmatrix with columns } \frac{\partial b_{p+1}^{(j)}{}^{(\mu)}}{\partial x_{i_{p+1}}} \qquad \text{(Jacobian)}$$

$$A_{p+1}^{(\mu)} J_{(\nu_2)}^{(\mu)} = B_{(\nu_2-1)}^{(\mu)} - Y_{\nu_2}^{(\mu)} - \sum_{j=1}^{\nu_2-1} \begin{bmatrix} \nu_2-1 \\ j \end{bmatrix} \left[\Omega_j^{(\mu)} + A_{p+1}^{(j)(\mu)} J_{(\nu_2-j)}^{(\mu)}\right] \qquad \nu_2=1,\dots,m_2$$

$$\left[I - \sum_{\nu_2=1}^{m_{2_i}} (-1)^{\nu_2+1} e_{\nu_2} h^{\nu_2} J_{(\nu_2)}^{(\mu)}\right] z^{(\mu)} + x_p + \sum_{\nu_1=1}^{m_{1_i}} d_{\nu_1} h^{\nu_1} x_p^{(\nu_1)} + \sum_{\nu_2=1}^{m_{2_i}} (-1)^{\nu_2+1} e_{\nu_2} h^{\nu_2} x_p^{(\nu_2)} = 0$$

$$x_{p+1}^{(\mu+1)} = x_{p+1}^{(\mu)} - z^{(\mu)}$$

The convergence of the Newton iteration requires that for $\eta_{p+1} = x_{p+1} - g_{p+1}$:

$$\left| \frac{\eta_{p+1} \dfrac{d^2\eta_{p+1}}{dx_{p+1}^2}}{\left[\dfrac{d\eta_{p+1}}{dx_{p+1}}\right]^2} \right| < 1 \qquad (3.24)$$

Therefore, the convergence can be controlled by the selection of the initial guess for $x_{p+1}^{(\mu)}$. If this initial $x_{p+1}^{(\mu)}$ is such that $\eta_{p+1} \approx 0$, then the convergence is nearly guaranteed to be quadratic.

Figure 3.11 indicates that the stability of the IPSE algorithm is hardly higher than the one of EPSE if an infinite number of fixed point iterations is used, while it is infinitely large if an infinite number of Newton iterations are used. If a finite number of iterations are used, for both type of iterations the stability region is closer to that of EPSE. The requirement to use the extremely time consuming Newton iteration (3.23) lowers the speed of the IPSE algorithm so much that even with the maximum step size of (3.1) it is much slower than the EPSE. Therefore, IPSE is not among the candidate algorithms.

Implicit Runge Kutta

The Implicit Runge Kutta (IRK) algorithm is based on implicit expressions for κs [Gear71]. It has accuracy and stability similar to IPSE, but it is even slower. It requires the iterative solution of m_i equations, one for each κ, instead of one equation for IPSE. Therefore, it is not considered as a candidate.

Implicit Multi-step

The most widely used Implicit Multi-step algorithms are the Adams Moulton (AM) and the Gear (GR) methods. The AM algorithms [Gear71] are formulated as:

$$x_{i_{p+1}} = x_{i_p} + h_i \sum_{j=p+1}^{p-m_i+2} \gamma_j \dot{x}_{i_j} \tag{3.25}$$

Provided that an infinite number of iterations has been applied to obtain the exact solution of (3.25), the parameter β, is shown in Figure 3.10. The stability region, for an infinite number of iterations is shown in Figure 3.6. Figures 3.10 and 3.11 indicate that the accuracy and stability of the AM algorithm is between those of the

EPSE and the AB algorithms. For a finite number of iterations its accuracy and stability is close to that of the AB algorithm. Being an implicit algorithm, AM requires a time consuming iterative solution of (3.25). The AB formula of the same order can be used to obtain the initial guess for x_{p+1}. Unlike the IPSE algorithm it can be shown that fixed point iteration suffices for the solution of (3.25). Even with fixed point iteration however, AM is still significantly slower per step than AB, while its accuracy and stability performance is hardly higher than the AB, as indicated by Figures 3.10 and 3.11. This apparent inferiority compared to the AB algorithm disqualifies the AM algorithm to be among the candidates.

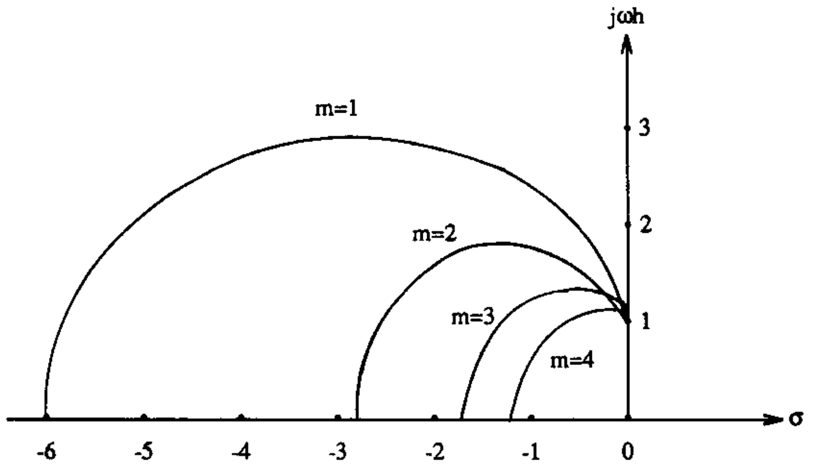


Figure 3.6: The stability region of the Adams Moulton algorithm
The algorithm is stable inside the curves

An intermediate solution to the low speed problem of the AM algorithm against the AB is the Adams Moulton Predictor Corrector (AMPC) algorithm. This corresponds to the AM with only one iteration of (3.25). Then an AB formula is used to "predict" the value of x_{p+1} and the AM formula is used to "correct" this prediction, as follows:

$$p_{i_{p+1}} = x_{i_p} + h_i \sum_{j=p}^{p-m_i+1} \varepsilon_j \dot{x}_{i_j} \tag{3.26}$$

$$x_{i_{p+1}} = x_{i_p} + h_i(\gamma_{p+1} p_{i_{p+1}} + \sum_{j=p}^{p-m_i+2} \gamma_j \dot{x}_{i_j})$$

Still (3.26) requires about twice as much computation as (3.19). Therefore, the AMPC algorithm is only half as fast as the AB algorithm. In addition, the β parameter, shown in Figure 3.10, indicates that the accuracy of the AMPC algorithm is lower than that of the AB. On the other hand, the stability region of the AMPC algorithm, shown in Figure 3.7, is a subset of the one shown in Figure 3.6. Figure 3.11 indicates that the stability performance of the AMPC algorithm is slightly better than that of the AB. Despite of the relatively close competition with AB, the AMPC is apparently inferior to the AB and cannot be among the candidate algorithms.

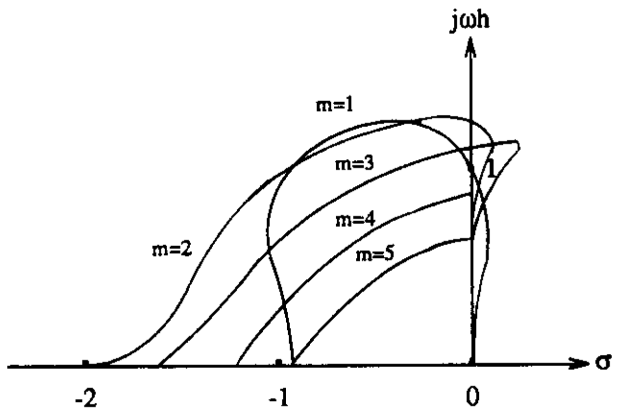


Figure 3.7: Stability of the Adams Moulton Predictor Corrector algorithm
The algorithm is stable inside the curves

The GR algorithm [Gear71] is based on the following backward differentiation formula (BDF):

$$x_{i_{p+1}} = \sum_{j=p}^{p-m_i+2} a_j x_{i_j} + h_i \gamma_{p+1} \dot{x}_{i_{p+1}} \tag{3.27}$$

An iterative solution of (3.27) is again required. The GR algorithm is widely used for stiff ODEs due to its very high stability. The β parameter, shown in Figure 3.10 for infinite number of iterations, indicates significantly lower accuracy than that of the EPSE. For an infinite number of iterations, the stability region is shown in Figure 3.8 and compared with the other algorithms in Figure 3.11. The formulas with order higher than 2 are unstable close to the imaginary axis and the formulas with order up to 2 are too stable as they would damp the low frequency components. This inappropriate stability region prevents the GR algorithms from being considered as candidates.

Implicit Multi-step Multi-derivative

The Implicit Multi-step Multi-derivative algorithms attempt to improve the stability region of the GR algorithm by introducing terms with higher derivatives. The Enright (ER) algorithm [Enri74, Chak83, Jelt77] is the most widely used. It is limited to the use of up to second derivative terms so that it does not become as time consuming as the IPSE algorithm. It is formulated as:

$$x_{i_{p+1}} = \sum_{j=p}^{p-k_i+2} a_j x_j + h_i \sum_{j=p+1}^{p-k_i+2} b_j \dot{x}_{i_j} + h_i^2 \sum_{j=p+1}^{p-k_i+2} c_j \ddot{x}_{i_j} \tag{3.28}$$

The order of this algorithm is k_i+2. For an infinite number of iterations, the β parameter, shown in Figure 3.10, indicates that its accuracy is lower than that of the EPSE. For an infinite number of iterations, the stability region is shown in Figure 3.9 and

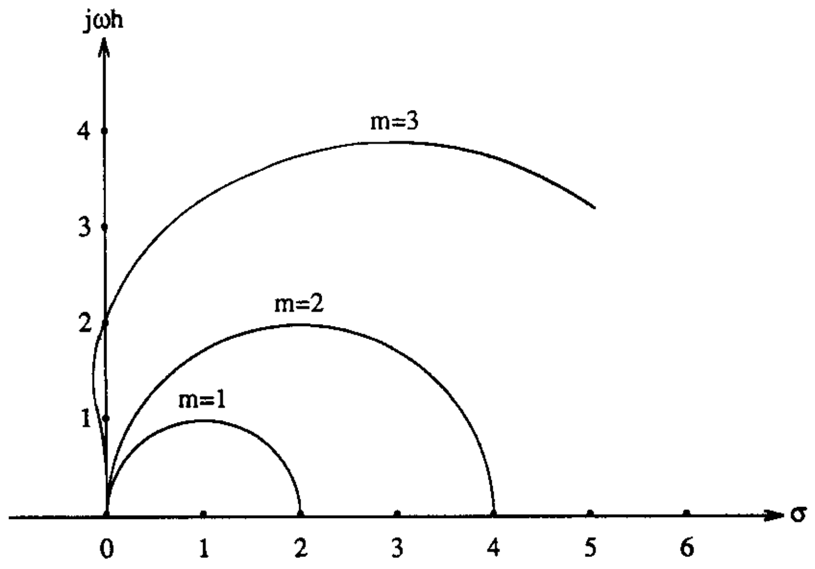


Figure 3.8: Stability region of the Gear algorithm
The algorithm is stable outside the curves

compared with the other algorithms in Figure 3.11. Apparently only the fourth order (k_i=2) formula has an appropriate stability region, despite the fact that it damps the components with frequencies only higher than $2/h_i > F_i$. The β parameter of the 4th order formula is approximately only 0.01. This limits the performance of the ER algorithm because even the largest step size defined in (3.1) cannot provide an acceptable accuracy as can be seen from (2.5). The ER algorithm is therefore not among the candidates.

Conclusion

Figures 3.10 and 3.11 compare the accuracy and the stability of the above discussed algorithms. Table 3.1 summarizes the performance of the algorithms. As can be seen from this table, the algorithms that are candidates to solve ODE systems

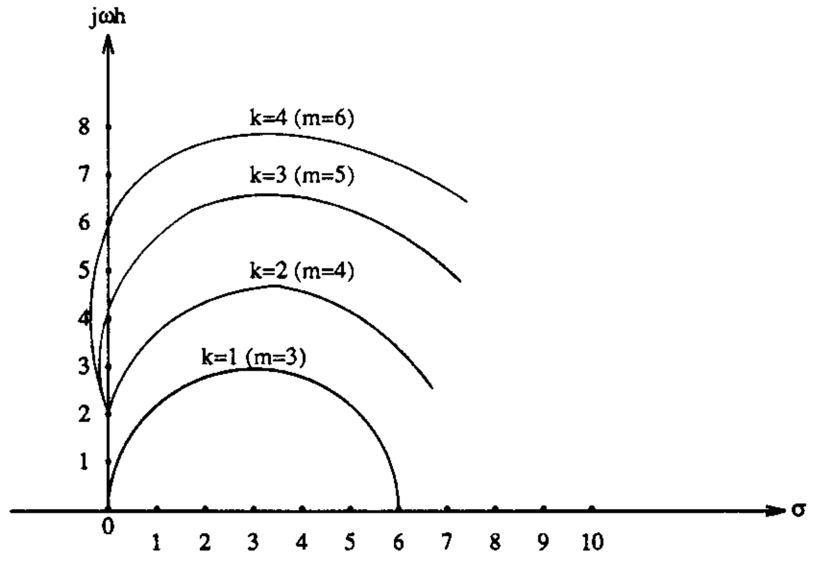


Figure 3.9: The stability region of the Enright algorithm
The algorithm is stable outside the curves

that appear in the mathematical models of the dynamics of space systems, are:

1. Explicit Power Series Expansion

2. Adams Bashforth

3.5 Further comparison of the candidate algorithms

The mathematical models of the dynamics of space structures may contain a number of functions of the state variables that include discontinuities, such as steps, hystereses, dead zone delimiters etc. Such functions can generally be modeled as:

$$f(x,t) = \begin{cases} f_1(x,t) & r(x,t) \geq 0 \\ f_2(x,t) & r(x,t) \leq 0 \end{cases} \tag{3.29}$$

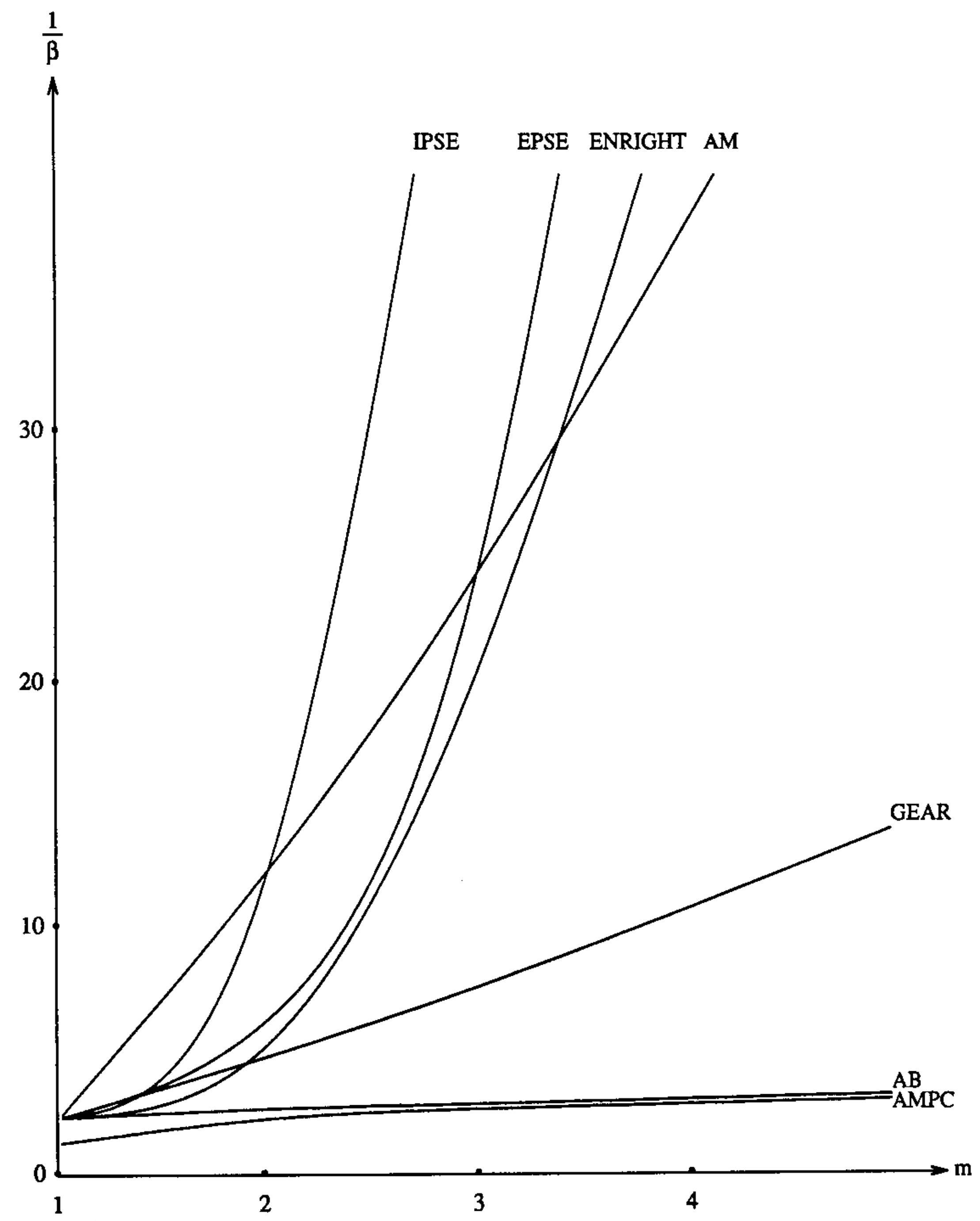


Figure 3.10: The inverse of the β parameters of the algorithms

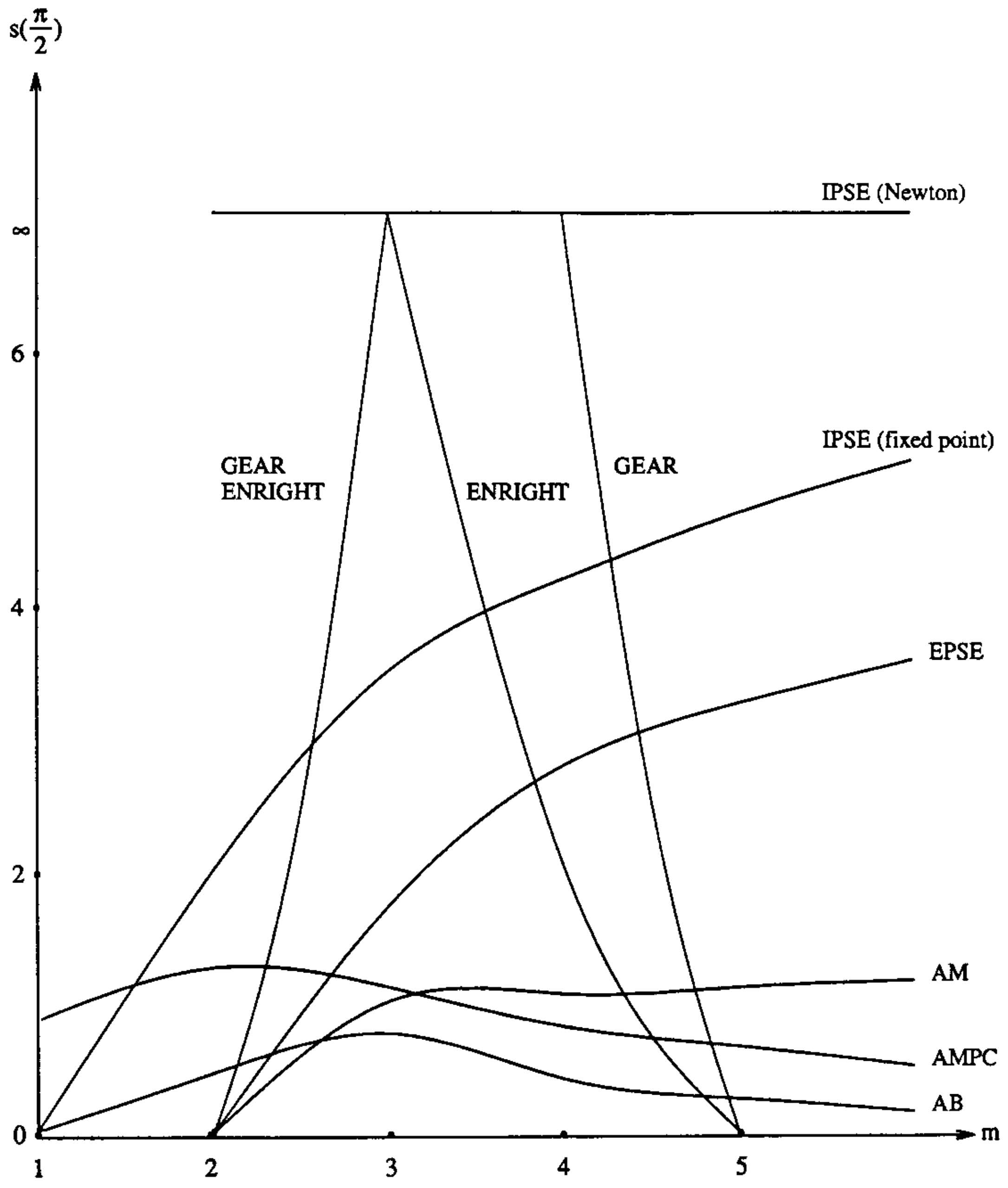


Figure 3.11: The stability region of the algorithms close to the imaginary axis

Table 3.1: Comparative performance of algorithms

ALGORITHM	SPEED PER STEP	ACCURACY	STABILITY	EXPECTED PERFORMANCE
Explicit Power Series Expansion	good	very good	very good	good
Explicit Runge Kutta	poor	very good	very good	poor
Adams Bashforth	very good	poor	poor	good
Implicit Power Series Expansion	very poor	very good	very good for $m_1=m_2$ but very poor else	very poor
Implicit Runge Kutta	very poor	very good	very good for $m_1=m_2$ but very poor else	very poor
Adams Moulton	very poor	good	medium	very poor
Adams Moulton Predictor Corrector	good	poor	poor	medium
Gear	poor	medium	very poor	very poor
Enright	very poor	good	good for m=4 but very poor else	very poor

These discontinuities may cause a discontinuity in the solution as shown in Figure 3.12.

Proceeding with a regular step size to compute x_{p+1} from x_p with $f(x,t) = f_1(x,t)$ would produce the erroneous result $\tilde{x}_{p+1}$. The accurate computation of the solution requires the solution at $t = \tau$ from x_p with $f(x,t) = f_1(x,t)$, followed by the computation of x_{p+1} from the solution at $t = \tau$ with $f(x,t) = f_2(x,t)$. The determination of τ requires iterative solution with different step sizes until a solution $\hat{x}$ at $t = \hat{\tau}$ is obtained so that $r(\hat{x},\hat{\tau}) \leq \varepsilon$, where ε is sufficiently small. Then τ can be approximated

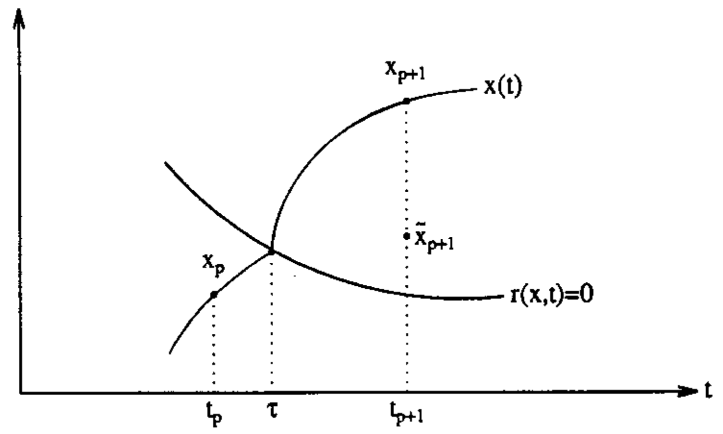


Figure 3.12: Solution across discontinuities

by $\hat{\tau}$. The repetitive solution can be carried out by doubling or halving the step size.

For the EPSE algorithm, the repetition of the solution does not cause a significant overhead. This overhead includes only the evaluation of the Taylor series (3.11) for different step sizes. Therefore, every repetition includes the evaluation of a polynomial with m terms.

For the AB algorithm however, as well as for any other multi-step algorithm, the repetition of the solution causes a serious overhead. Each repetition includes the computation of the solution of m+1 points of the solution, i.e. m past points, which are no longer available, and the new point. To reduce this overhead, first order formulas can be used. Then no past points are required. It should be realized however, that first order formulas provide less accuracy. In addition, the computation of the first m points after the solution at $t = \tau$, requires the use of formulas of orders that start from 1 for x_{p+1} and advance by one for every subsequent point until order m for

x_{p+m}. Therefore, the accuracy in computing these m points would be reduced.

It should be also recognized that the other algorithms, such as the explicit Runge Kutta or any of the implicit ones, require an enormous overhead for computing the solution across discontinuities. The superiority of the EPSE algorithm in computing the solution across discontinuities is a very important advantage for this algorithm in the solution of ODEs that appear in the models of the dynamics of space structures where the discontinuities are usually frequent.

Although the models of dynamics of space systems are not expected to have complicating features other than discontinuities, it is important to mention that the superiority of the EPSE algorithm applies to other types of peculiarities, such as singularities and time delays. It is also important to realize that the ESPE algorithm is more more difficult to implement and requires more storage.

The EPSE algorithm has additional advantages compared to the AB algorithm, when implemented in a multi-rate scheme. As explained in Section 2.2, in a multirate implementation of an algorithm different step sizes and orders are used for different groups of ODEs. The following section suggests multirate implementations for each of the candidate algorithms and shows the advantages of the EPSE algorithm.

3.6 Multirate implementation of the candidate algorithms

In most cases, the frequency content of the state variables may be quite different. Higher frequencies may affect significantly only state variables that correspond to flexible bodies and bodies of small mass. Depending on their frequency content, the ODEs of a system can be grouped, as shown in Figure 2.3, so that the ODEs in each group require the same means for their solution. Different step sizes and orders can be used for different groups of ODEs to increase the speed of the solution of the entire ODE system. Then the algorithm to be used has to be implemented in a multirate scheme, that is, it will solve the different groups of ODEs in parallel but with different rates.

Let us explain the multirate implementation of algorithms with the aid of an example. Also consider for the moment the use of different step sizes. Assume that a system of N ODEs is to be partitioned into three groups of N_1, N_2 and N_3 ODEs respectively. A typical reason for such a decision can be the fact that N_1 of the ODEs are expected, after the frequency analysis shown in Figure 2.3, to have significant frequency content up to F Hz, N_2 of the ODEs up to $\frac{F}{5}$ Hz, and N_3 of the ODEs up to $\frac{F}{20}$ Hz. Then the maximum step size that can be used for each of the groups is $h_1 = h$, $h_2 = 5h$ and $h_3 = 20h$ respectively. Figure 3.13 shows the points of the solution that are to be computed for each of the three groups. The sequence of the computation of points of the solution that is shown in Figure 3.13 is periodically repeated. The largest step size $h_3 = 20h$ defines the period. It should be recognized that at every step different groups of ODEs are to be solved. The following three cases appear:

1. The next point of the group #1 of N_1 ODEs is to be computed with step size h_1.

2. The next point of the group #1 of N_1 ODEs and of the group #2 of N_2 ODEs is to be computed with step size h_1 and h_2 respectively.

3. The next point of the group #1 of N_1 ODEs, of the group #2 of N_2 ODEs and

of the group #3 of N_3 ODEs is to be computed with step size h_1, h_2 and h_3 respectively.

Each of the solution points in Figure 3.13 is marked according to which case it belongs. Due to the relative magnitude of the step sizes, the first case appears 80% of the cases, the second case 16% and the third case 4%.

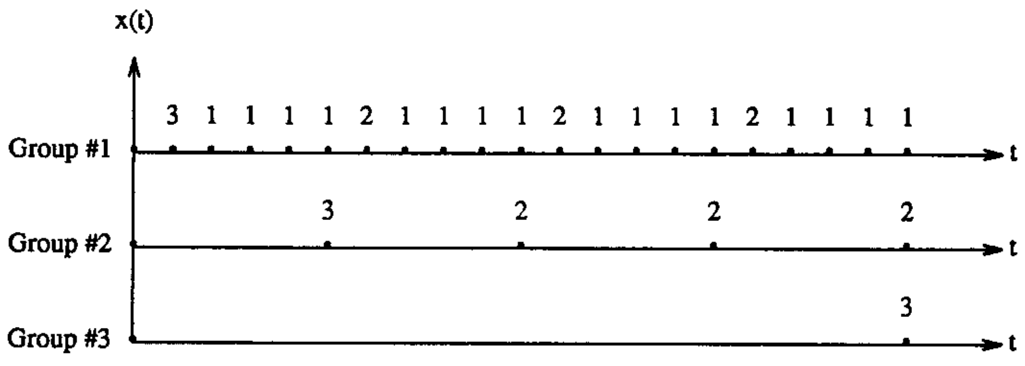


Figure 3.13: Multirate solution of an ODE system

The computation of the next solution point requires the availability of the solution at past points. For algorithms that require the computation of high derivatives, such as the EPSE, the derivatives at past points are needed too for the computation of the next solution point. For most of the points however, not all the needed past points or derivatives would be available. The computation of the solution at $t=2h_1$ of the ODEs in group #1 for example, needs the solution and the derivatives for the ODEs of group #2 and #3 at $t=h_1$, which are not available. Points that are not available can be approximated by an interpolation between available points. Derivatives that are not available can be approximated with finite differences. The order of the interpolation and finite difference formulas can be selected according to the desirable accuracy. Linear interpolations and first order finite differences can be used to minimize the overhead.

The impact of the multirate implementation of algorithms is that significant part of the computation to obtain the next point in the solution, for 84% of the points for the ODEs in group #2 and 96% of the points for the ODEs in group #3, is skipped or substituted by relatively much less time consuming interpolation and evaluation of finite differences.

According to (3.19), the computation of each solution point, say *p*, with the AB algorithm requires:

1. The evaluation of the elements of the matrix A_p and the vector b_p.

2. The evaluation of the elements of the vector $\dot{x}_p$ from the solution of the linear system $A_p\dot{x}_p=b_p$.

3. The evaluation of the elements of the vector x_{p+1} from (3.19).

4. The evaluation of the elements of the vector x at $t=kh_1$ by interpolation.

Figure 3.14 shows the parts of the mathematical model that must be processed for each of the three solution cases mentioned above. For the sake of clarity of the Figure, it is assumed that the first N_1 rows constitute the group #1, the following N_2 the group #2 and the last N_3 the group #3.

Figure 3.15 shows the multirate implementation of the AB algorithm, and particularly the steps that have to be taken during each period of the solution. Each step involves the computation of one of the three cases, marked as I, II and III

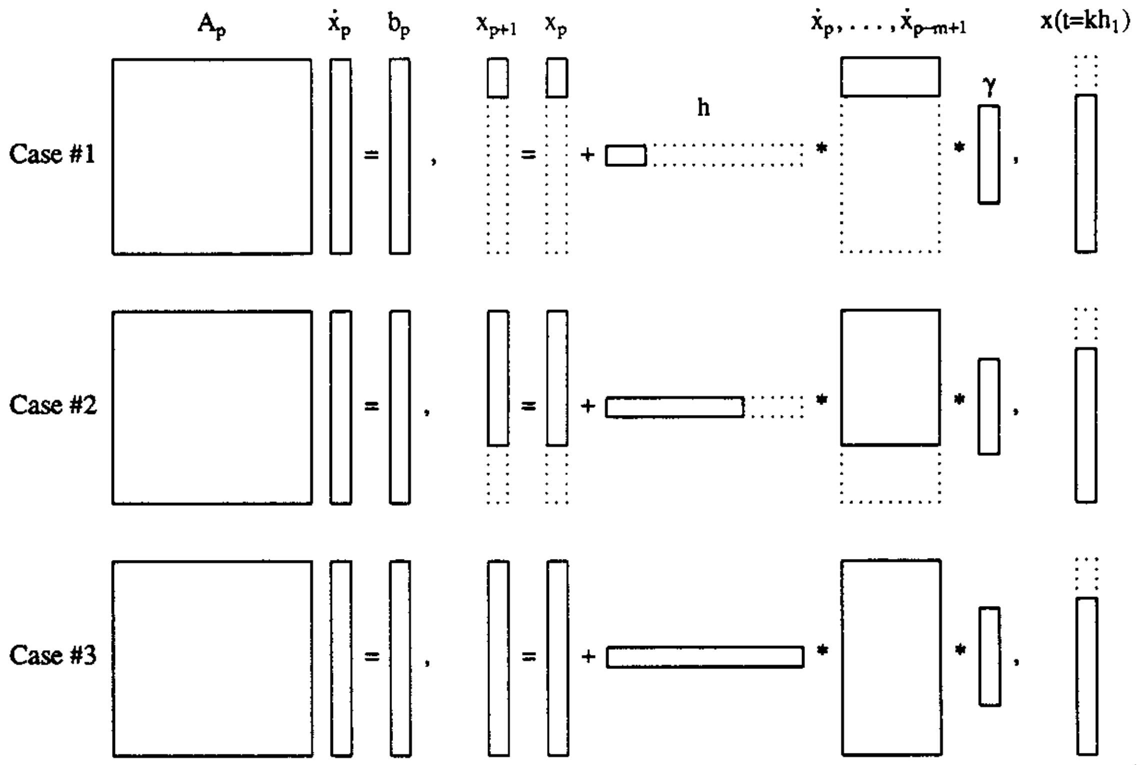


Figure 3.14: Multirate AB algorithm

respectively. The meaning of the three symbols is the following:

1. The circles represent a full computation of the formula of the algorithm. The process of group #1 of case #1 is an example.

2. The crossed circles represent a full computation of the formula followed by an interpolation. The process of group #2 of case #2 is an example.

3. The squares represent a partial computation of the formula due to the savings explained in Figure 3.12. The process of group #2 of case #1 is an example.

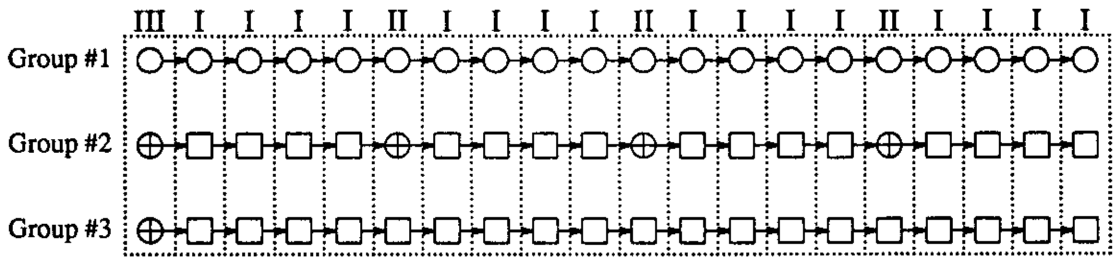


Figure 3.15: The implementation of multirate algorithms

According to (3.14) and (3.16), the computation of each solution point, with the EPSE algorithm requires:

1. The evaluation of the elements of the matrix A_p and the vector b_p.

2. The evaluation of the elements of the vector $\dot{x}_p$ from the solution of the linear system $A_p\dot{x}_p=b_p$.

3. The evaluation of the elements of each of the m derivatives, say the νth derivative, of the vector x_p according to the following steps:

   i. Evaluation of the νth derivative of the elements of the matrix A_p and the vector b_p.

   ii. Evaluation of the vector z_{ν_p} either from (3.14) or from (3.15), depending on the group. It can be shown that the evaluation from (3.15) reduces to $z_{\nu_p} = b_p^{(\nu-1)}$

   iii. Evaluation of the νth derivative of the elements of the vector x_p from the solution of the linear system $A_p x_p^{(\nu)}=z_{\nu_p}$.

   or by finite differences, depending on the group.

4. The evaluation of the elements of the vector x_{p+1} from (3.16).

5. The evaluation of the elements of the vector x at $t=kh_1$ by interpolation.

Figure 3.16 shows the parts of the mathematical model that must be processed for each of the three solution cases mentioned above. Figure 3.15 can express the multirate implementation of the EPSE as well. The boxes represent a computation that evaluates the vector z from (3.15) and evaluates the derivatives of the vector x by finite differences.

Figures 3.14 and 3.16 indicate that the computation that is saved for the EPSE algorithm is much more significant than the AB algorithm because the evaluation of the elements of the matrix A, the vector b and their derivatives is the most time consuming part of the solution process.

In addition to different step sizes, the groups of ODEs may need to be solved with formulas of different order. This does not create any additional complications for the AB algorithm. If the EPSE algorithm is to be used however, then a slight modification of the cases #2 and #3, as shown in Figure 3.16, has to be introduced. Let us assume that the relation between the orders to be used for the three groups is $m_1>m_2>m_3$. Then in case #2 for example, the computation of the vectors z_v and $x^{(v)}$ has to switch to (3.15) and finite differences respectively for $m_2<v\leq m_1$. This is equivalent to switching from case #2 to case #1. Similarly for case #3 one has to switch to case #2 for $m_3<v\leq m_2$ and to case #1 for $m_2<v\leq m_1$. The computational savings offered by this switching requirement are much more significant for the EPSE algorithm than for the other candidates.

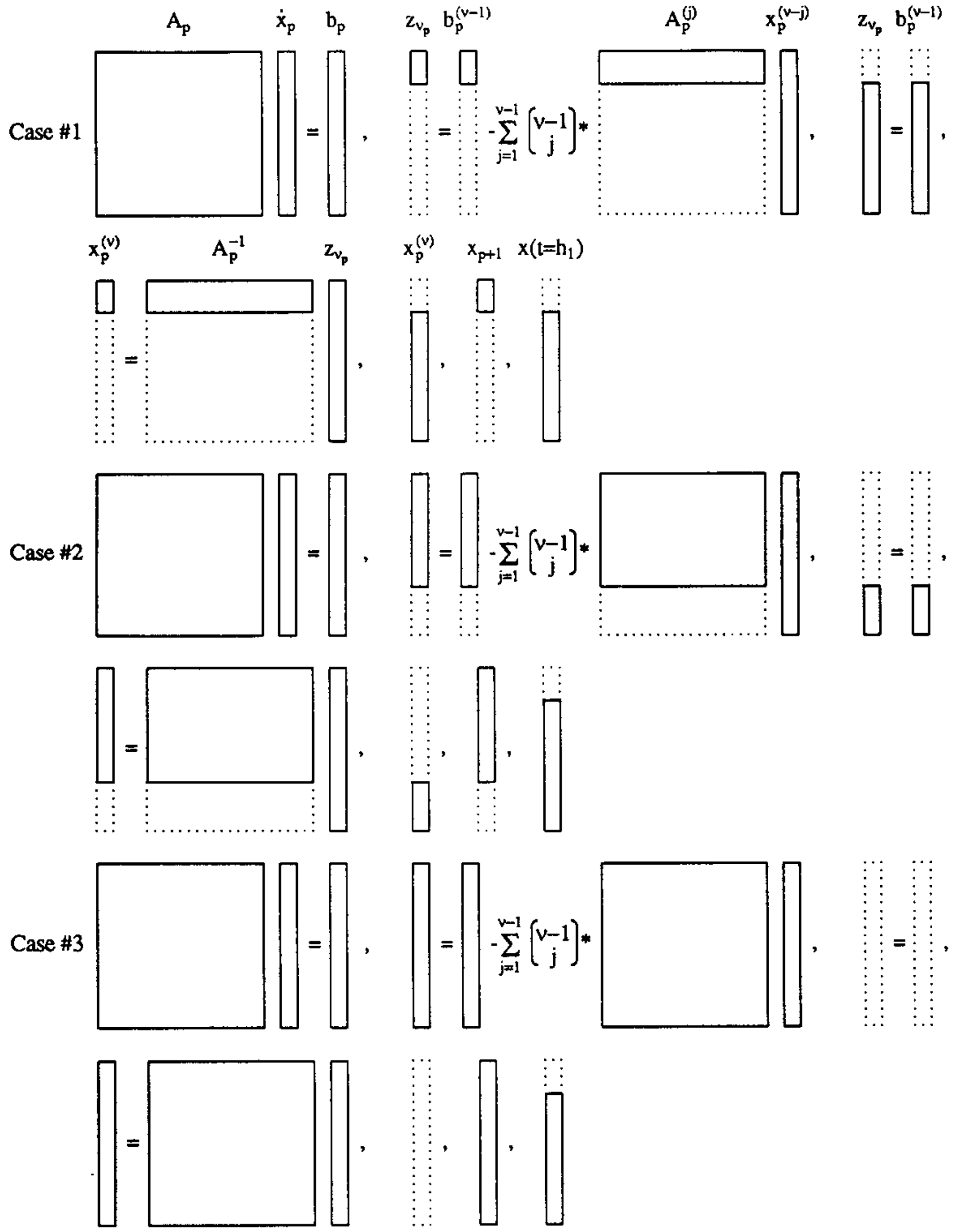


Figure 3.16: Multirate EPSE algorithm

Finally it is important to realize that a reduction of the accuracy and the stability of the algorithms is a penalty to pay for the speedup of the solution that the multi-rate implementation offers. The magnitude of this reduction is usually small enough however, so that the multirate implementation of algorithms is in general recommended.

3.7 Summary

Chapter 2 introduced in a high level the procedure for the selection of algorithms to solve ODE systems. The procedure in its high level is applicable to any type of ODEs. This chapter suggests the implementation of the high level modules in the case of the specific ODE systems that appear in the space structures simulation. The selection of the algorithms itself appears in the next two chapters.

An analysis of the computational characteristics of the ODE systems that appear in the models of space structures shows that these ODE systems are large consisting of several hundreds of ODEs, highly nonlinear including trigonometric functions and discontinuities, highly oscillating with a range of frequencies of 0.5 - 100 Hz, lightly damped taking around 300 secs to reach steady state, and in semiexplicit form $A(x,t)\dot{x}=b(x,u,t)$. An analysis of the requirements of the solution of the ODE systems shows that real time solution may require a large number of processors, while an accuracy corresponding to up to 1% error is desirable. A fairly small step size defined by (3.1) is required to represent the high frequency components accurately. The slow damping of the oscillations together with the need for small step size require that a large number of points of the solution is computed. Then a very stable algorithm is needed to achieve acceptable accuracy. The desirable stability region is shown in Figure 3.1. The implementation of the accuracy estimation module of the algorithm selection procedure, and in particular of the part shown in Figure 2.4, is suggested to be based on (2.5) and (3.5). The speed, accuracy and stability bounds of the procedure are specified in (3.6) and Figure 3.1. A qualitative performance comparison of a set of the most widely used algorithms is performed. Table 3.1 summarizes the results of this comparison. These results suggest the candidate algorithms to be further compared using the algorithm selection procedure. The Explicit Power Series Expansion and the Adams Bashforth algorithm are finally included among the candidates. It is realized that in most cases the algorithms would have to be implemented in a multirate scheme. The last section of the chapter includes suggestions for the multirate implementation of the candidate algorithms. Overall it is expected that Explicit Power Series Expansion algorithm would be the most advantageous in most cases. It is shown to be far superior in computing the solution across discontinuities. It also exploits the multirate scheme significantly more than the other candidates.

4. MULTIPROCESSOR IMPLEMENTATION OF ALGORITHMS

Chapters 2 and 3 present a procedure that automatically selects algorithms to solve ODE systems that appear in the models of the dynamics of space structures. This chapter proposes a procedure for the implementation of the algorithms on multiprocessor computers. The first section defines the problem of the implementation and the criteria for its optimization. The second section presents the procedure defining its components in a high level. The following 5 sections are devoted to the detailed presentation of each of the modules of the procedure. These modules include the partition of the computation into tasks, the selection of the optimal number of processors, the assignment of the tasks to the processors, the sequencing of the execution of the tasks on each processor, and the iterative improvement of the implementation.

4.1 The objective of the procedure

The procedure for the multiprocessor implementation of algorithms that solve ODE systems is an important part of the application software generator of multiprocessor computers with architectures like those shown in Figures 1.2 and 1.3. It has to optimize the utilization of the hardware power of such multiprocessor computers in solving ODE systems that appear in the models of the dynamics of space structures. The multiprocessor implementation of algorithms is still an open distributed processing problem. No standard software or even techniques exist yet, despite of the great demand. The problem is too broad to be solved in full generality. There are many kinds of algorithms, of formulation of ODE systems and of multiprocessor architectures. The procedure that is proposed in this dissertation entails the following assumptions:

1. The ODE system is of the form of $A(x,t)\dot{x}(t)=b(x,u,t)$.

2. The multi-rate EPSE or AB algorithms are employed, since they were shown in Chapters 3 to be the most advantageous one for solving the ODE systems of interest.

3. The processors of the multiprocessor computer include I/O units that function independently from their ALUs allowing the interprocessor communication to be performed simultaneously with the arithmetic operations.

4. The interprocessor communication takes relatively short time.

The procedure is sufficiently general so that it can be easily modified to apply to different multiprocessor architectures, different formulation of the ODE systems, and different algorithms.

The procedure can suggest either static or dynamic implementations, involving off line or on line decisions respectively. A dynamic implementation is more general and easier to develop, but it may cause a serious overhead. This overhead can be the result of delays due to the need to monitor the status of the processors or due to the need for more interprocessor and processor-host communication. A 50-100% overhead would not be surprising. In addition, a dynamic implementation cannot be optimal because an on-line prediction of the characteristics of the computation is practically impossible. It is apparent that the procedure should be static.

As explained in Chapter 1, the procedure to be developed has to meet the following criteria:

1. Maximization of the speed of the solution of the ODE system.

2. Minimization of the simulation cost: This can be accomplished by minimizing the hardware required and by having a simple and fast procedure.

According to (2.2), the solution speed can be maximized by:

1. Maximization of the step size used for the solution.

2. Minimization of the time to compute one point of the solution. This can be achieved by the:

    a. Minimization of the computational load involved in the solution.

    b. Optimization of the distribution of this minimized computational load among the processors.

    c. Minimization of the interprocessor communication. This involves:

        i. Execution of I/O operations, including communication protocols.

        ii. Transferring data among processors.

        iii. Time during which processors remain idle because they are awaiting data from other processors.

The maximization of the step size and the minimization of the computational load are part of of the algorithm selection procedure. The contribution of the multiprocessor implementation of algorithms to the maximization of the solution speed could consist of:

1. Balanced distribution of the computational load among the processors.

2. Minimization of interprocessor communication.

It should be recognized that the execution of I/O operations can be overlapped with the execution of arithmetic operations. In addition, the time that is required to transfer data between processors depends on the power of the communication media, which can be either a shared memory or a communication network. The minimization of this time is a hardware design issue not related to the optimality of the multiprocessor implementation procedure. As a result, the maximum solution speed criterion suggests a computational load distribution which achieves:

1. Balanced computational load of the processors.

2. Minimum idle time of the processors.

The achievement of this combination requires in sequence:

1. Optimal partitioning of the computation involved in the solution of the ODE system into a number of computational tasks.

2. Optimal assignment of these tasks to the processors.

3. Optimal sequencing of the execution of the tasks that have been assigned to each processor.

In addition to maximizing the solution speed, the procedure should be simple and fast. The simplicity criterion refers to the consideration of developing a procedure that is easy to implement and modular enough to be easy to modify and debug. The criterion of fast procedure refers to the consideration of developing a procedure that is the least time consuming possible to achieve a desirable solution speed.

Finally the impact of the minimum hardware criterion is the consideration of minimizing the number of processors and the storage needed, to achieve a desirable speed. The development of the procedure is directed, on one hand to taking maximum advantage of available hardware, and on the other hand to indicating the number of processors which are required for optimal operation.

4.2 The multiprocessor implementation procedure

The complexity of the multiprocessor implementation of an algorithm that solves an ODE system depends on the structure of the computation involved. The computation of the entire solution consists of the sum of the computations of the individual points of the solution. The structure of the computation is very similar for each solution point. Only special cases, such as the solution across discontinuities, require computations of different structure. Still the differences however, are not major. The implementation of an algorithm in regular cases can clearly imply the implementation in special cases. It is therefore necessary, to optimize the multiprocessor implementation of an algorithm based only on the computation involved in the evaluation of a typical point of the solution.

The computation involved in the evaluation of one point of the solution of an ODE system can be represented in a computational graph. Such graphs, called precedence graphs, show the precedence relations between different parts of the computation. The representation can be in many levels of complexity. At one extreme, one can have a trivial graph with one node representing the entire computation. In another extreme, one can have a very large graph with nodes representing elementary operations. Figure 4.1 shows the graph at the level of elements of the matrices and vectors of an ODE system in the form $A(x,t)\dot{x} = b(x,u,t)$, when it is solved using the single-rate EPSE algorithm according to (3.14) and (3.16). Note that the graph has the pattern of an array with N columns, one for each of the N state variables, and m rows, one for each of the m derivatives of the state variables. Each element of the array includes the computation of one derivative of a state variable. The structure of the computation for all the derivatives is the same with the exception of the first derivative, which includes the computation required for the inversion of the matrix A.

For the multi-rate EPSE algorithm, the computation involved in the evaluation of one point of the solution in each of the solution cases can be represented by a graph similar to the one in Figure 4.1. Referring back to Figures 3.13 and 3.14, note that the computations marked with a circle in Figure 3.13 can be represented by one of the columns of the graph in Figure 4.1. The computations marked with a crossed circle can be represented in the same way, including however, the computation of an interpolation formula in the evaluation of x_{p+1}, shown in the bottom nodes. The computations marked with a square can be represented by a subset of a column of the graph. The second column of the graph in Figure 4.2 shows such a computation. The nodes that represent the evaluation of the vector z_v and the vector $x^{(v)}$ are based on (3.15) and finite differences respectively. As a result, the graph of the cases #1 and #2 in Figure 3.13 can be represented by Figure 4.2, while the graph of the case #3 can be represented by Figure 4.1. In the case of different orders for the different groups of ODEs, there will be columns that combine the upper nodes of the first column with the lower nodes of the second column in Figure 4.2. It is apparent that in every case, the array structure of the graph is preserved in the multi-rate implementation of the EPSE algorithm.

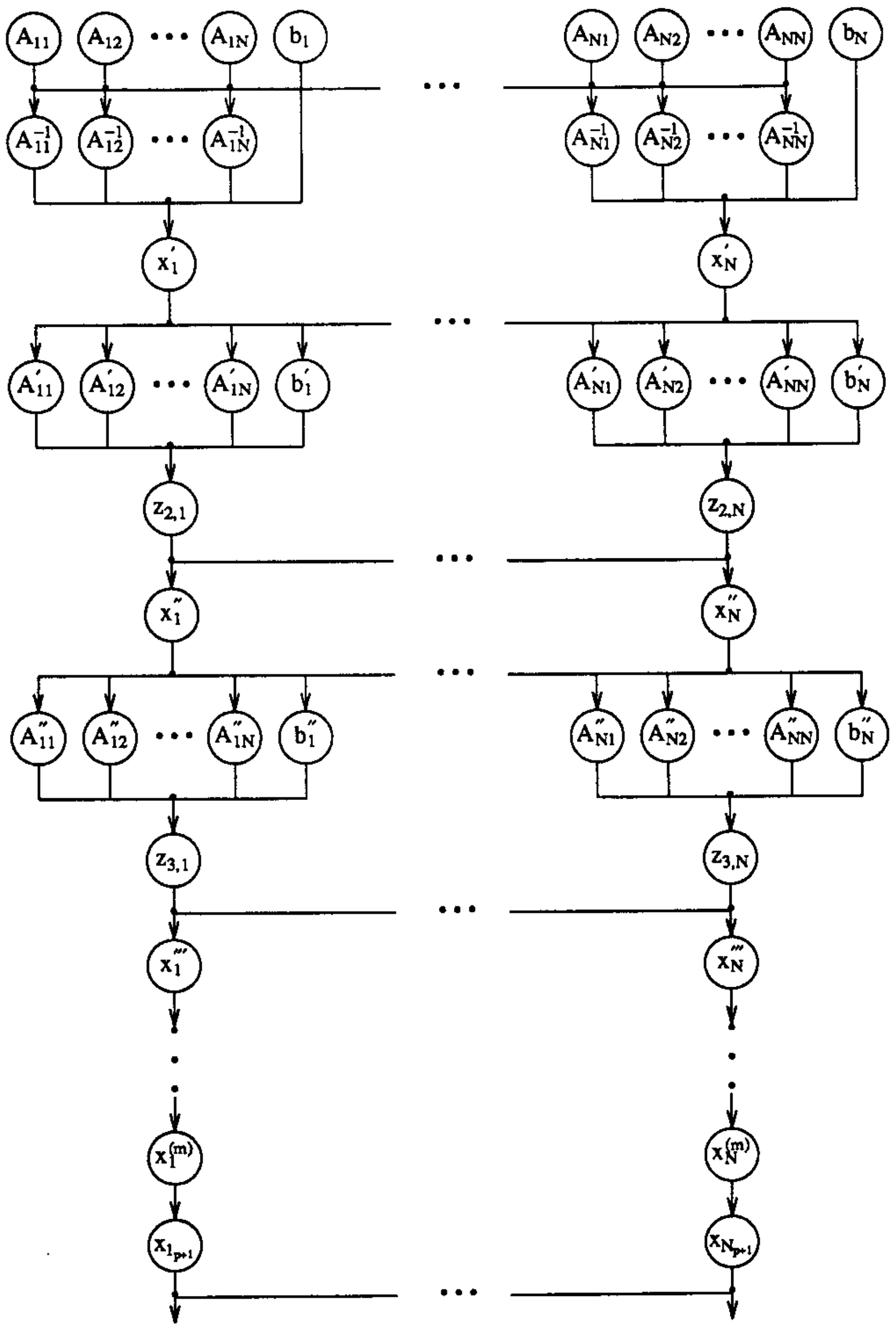


Figure 4.1: Graph of the solution of $A\dot{x}=b$ with the single-rate EPSE algorithm

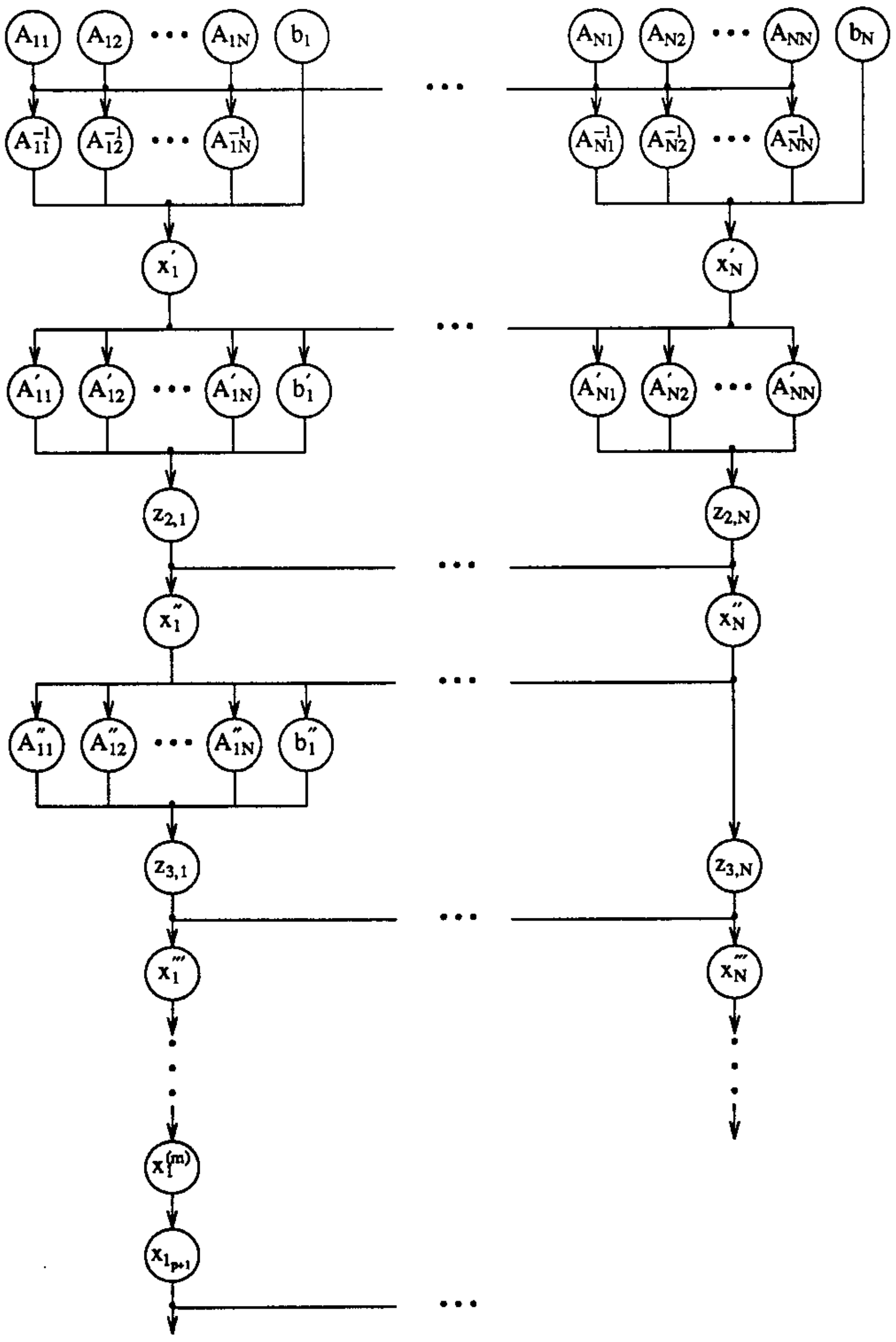


Figure 4.2: Graph of the solution of $A\dot{x}=b$ with multi-rate EPSE algorithm

The structure of the graph of any other algorithm follows closely the array pattern as well. All graphs have N columns but differ in the number of rows. Multi-step algorithms do not include the rows corresponding to the evaluation of the higher derivatives. Implicit methods repeat the rows of the corresponding explicit ones. The computational graph for the AB algorithm for example, is a subset of the one shown in Figure 4.1. It does not include the computation below the evaluation of the first derivative of x with the exception of the evaluation of the vector x_{p+1}.

The determination of an optimal multiprocessor implementation of an algorithm is a typical resource allocation or scheduling problem, which requires the optimal schedule of the execution of a number of nodes of a graph by a number of processors. In the present case:

1. The graph is deterministic, i.e. all the information needed about it is known.
2. The graph is acyclic, i.e. it does not have any loops or cycles.
3. The graph is unconditional, i.e. it does not have any decision nodes.
4. The graph has a general precedence structure, i.e. each node can have an arbitrary number of predecessors an successors.
5. The times of the execution of the nodes are of unequal length.
6. The number of processors is not limited to two.
7. The goals of optimization are:
    i. Minimization of the completion time ω, which is the time to finish the execution of all the nodes.

ii. Achievement of a completion time that is not later than a deadline τ_u specified by the bound on the solution time S_u.

iii. Minimization of the number of processors that are required to meet the deadline.

It is also assumed that:

1. The processors are identical.

2. Unlimited resources, such as storage or interprocessor communication paths, are available.

3. The interprocessor communication delay is short and constant. As soon as a datum to be transferred becomes available, it is assumed to become also available to any processor that is expecting it in a short constant time. This may not be a realistic assumption for some computer systems. Variations of the communication delay can be introduced by the I/O mechanism or by the communication protocols, including routing and retransmission in case of error. The assumption simplifies the resource allocation problem however, because it is very difficult in general to model variable communication delays deterministically. The capability of processors to have their I/O operations overlapped with the arithmetic reduces the harmful impacts of such an assumption.

4. No preemption takes place, that is, a node cannot be interrupted to execute another node of higher priority. Preemption may increase the execution speed but is more difficult to implement.

5. No deadlines exist for individual nodes or clusters of nodes.

The great modularity of the procedure that is proposed in this chapter provides enough flexibility to allow a user desiring to improve the procedure and willing to pay the price of higher complexity, to insert different modules, that employ heterogeneous processors, limited resources, preemption or individual node deadlines.

Such resource allocation or scheduling problems are known to be NP-complete [Coff75, Gare79]. Their optimal solution cannot be obtained in polynomial time $O(N_N^k)$ but rather in exponential time $O(M^{N_N})$, where N_N is the number of nodes and M is the number of processors. This is because all the different combinations of nodes and processors have to be compared. For a relatively small problem of 100 ODEs, a graph representation of the level shown in Figure 4.1 would result in 120,000 nodes. The number of different combinations to be compared for a computer with 4 processors is of the order of 10^{68000}.

Low complexity heuristic methods have been developed to solve scheduling problems. In [Gonz77] one can find an excellent survey of scheduling heuristics. Heuristics do not in general provide optimal solutions, i.e. minimum completion time or minimum number of processors, but attempt to provide near optimal solutions. The performance of a heuristic depends on the particular graph. It can be measured in terms of the proximity of the heuristic solution for the particular graph, ω and M, to the optimal one, ω_o and M_o. The following percentage measures are usually considered:

$$P = 100\frac{\omega - \omega_o}{\omega_o}\% \quad \text{and} \quad P = 100\frac{\left|M - M_o\right|}{M_o}\% \tag{4.1}$$

Worst or average case performance can establish a heuristic method. It should be recognized, that there can always be a best case graph that a heuristic is optimal. In developing a heuristic one should realize that lower complexity methods are usually less optimal. It should also be mentioned that the analytic determination of the performance or the complexity of many heuristics is practically impossible.

Among the several heuristics that could apply to our case, the most widely accepted assign individual nodes to a given number of processors by trying to limit the total execution time to the time to execute the nodes on the longest path of the graph [Kohl75, Ravi86], termed the critical path. This is because, the time to execute sequentially the nodes on the longest path is the minimum possible time to execute the entire graph. In [Fern73] one can also find a bound on the number of processors required to prevent the completion time from exceeding the longest path. The computational complexity of the most efficient of these heuristics is $O(N_N \log_2 N_N)$ to $O(N_N^3)$, depending on their optimality. For a relatively small problem with 100 ODEs, 4 processors and a graph representation of the level of Figure 4.1, these heuristics would take 10^7 to 10^{16} steps. Although this is not unacceptably long, many users would like to have a heuristic that is linear or logarithmic or at most low order polynomial with the number N of ODEs, rather than with the number of nodes N_N, so that the results are delivered promptly. The procedure that is proposed in this chapter attempts to serve such a purpose and follows a different approach. To reduce the computational complexity involved, the procedure suggests the partitioning of the graph into a number of clusters of nodes. This results in a graph with a smaller number of nodes N_N. In particular, as is explained in the following section, it suggests a partitioning that results in a number of clusters of nodes that is of the order of the number of ODEs of the system.

The procedure is based on a heuristic approach specialized for the computation shown in Figure 4.1. As is the case for all heuristics, the procedure cannot guarantee an optimal solution, i.e. maximum speed for the solution of the ODE system or minimum number of processors. It rather attempts to approximate closely enough the optimal solution. The procedure does however, allow the asymptotic approach to the optimal solution speed. The user can specify the bound on the solution speed that the procedure should achieve. The closer to maximum the desirable solution speed is, the more time does it take the procedure to achieve it.

To summarize, the procedure for the multiprocessor implementation of multi-rate algorithms should suggest for each solution case the optimal:

1. Partition of the computational graph into a number of clusters of nodes, called tasks.

2. Number of processors.

3. Assignment for execution of the tasks to the processors.

4. Sequence of the execution of the tasks on each processor.

Figure 4.3 shows the modules of the procedure at a high level. The following 5 sections in this chapter are devoted to the detailed description and suggestion for the implementation of these modules.

Finally it should be recognized, that an optimal schedule should in fact try to exploit all the kinds and levels of parallelism that appear in the computation. To exploit parallelism that might exist at levels lower than the one shown in Figure 4.1, a larger more detailed graph would be needed. Then the time to obtain an optimal

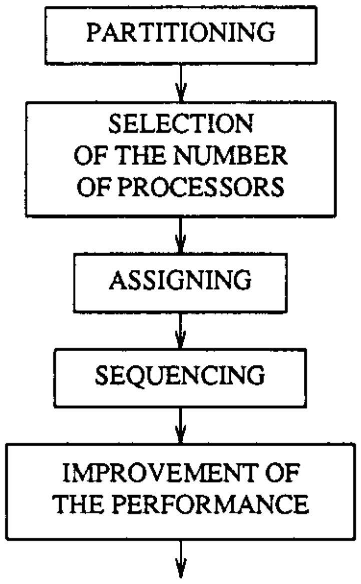

PARTITIONING
SELECTION OF THE NUMBER OF PROCESSORS
ASSIGNING
SEQUENCING
IMPROVEMENT OF THE PERFORMANCE

Figure 4.3: High level components of the multiprocessor implementation procedure

allocation would be even longer. Optimizing the schedule of the execution of the nodes of a graph at a high level, such as the one in Figure 4.1, constitutes a compromise between absolute optimality and complexity. The same compromise applies in optimizing the multiprocessor implementation of an algorithm over the computation involved in the evaluation of one point of the solution, rather than the entire solution, as mentioned in the beginning of this section.

4.3 The partitioning module

In order to keep the multiprocessor implementation procedure from becoming excessively complex and time consuming, it appears reasonable to partition the computational graph of Figure 4.1 or 4.2 into a small number of large autonomous clusters of nodes, called tasks, that are tightly coupled to each other and relatively loosely coupled to the nodes of the other clusters. This is because, the computational complexity of the assigning and sequencing procedures depends directly on the number of the tasks that the graph has been partitioned and the complexity of their interdependence. In addition, the minimization of the task interdependence results in minimum storage requirements. This is because, the data needed to be duplicated in different processors is minimized. It is important to realize, that the introduction of clustering does not absolutely lead to maximum integration speed, but constitutes a compromise. It sacrifices a small amount of integration speed for simplicity and for economy of storage.

In order to achieve high integration speed it is necessary to satisfy simultaneously two goals:

1. The partitioning of the graph into a number of tasks with minimum interdependence. This aims to minimize the idle time of the processors.

2. The partitioning of the graph into a number of tasks with such execution time lengths so that the optimal assignment of these tasks to the processors results in a balanced distribution of the overall computation.

In the graph shown in Figure 4.1 it is easy to observe the the nodes in each column are tightly coupled to each other and loosely coupled to the nodes in the other columns. For each state variable, say x_k, the nodes that constitute the kth column of the graph correspond to the processing of the elements of the kth row of the ODE system. Therefore, the nodes that represent the operations involved in the processing of each one of the rows of the ODE system seem to be the best candidates to be clustered together and assigned to the same processor. For multi-rate algorithms, the computational graph of each solution case has to be partitioned into tasks.

As each case contains all the rows of the previous cases, it appears reasonable to apply the partitioning on the entire ODE system, or equivalently on the last case, rather than on individual cases separately.

This partitioning results in only N loosely coupled tasks. It is very simple but sufficient. It may occur occasionally however,that one or more of the tasks is longer than the deadline τ_u allowed by the bound S_u on the speed of the solution of the ODE system. Then, even an optimal assignment would result in a load for some processors that violate the speed constraint, as shown in Figure 4.4.

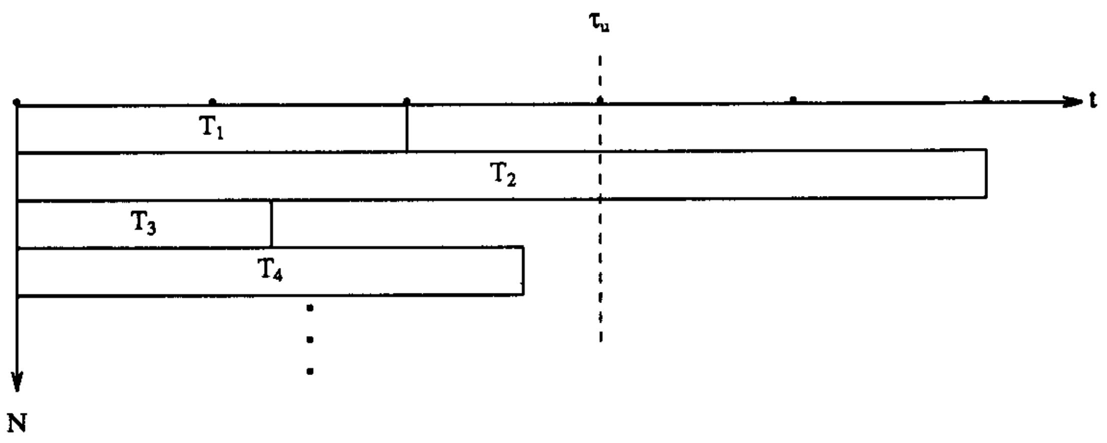


Figure 4.4: A long task that violates the speed constrain

In that event it is feasible to partition each of the long tasks into a number of subtasks. If a task is of length $L\tau_u+\tau_o$ where $\tau_o < \tau_u$ and L is a positive integer, then the task should be partitioned into L+1 tasks. The first L tasks should be of the maximum possible length that does not exceed τ_u. The remainder should constitute the last subtask. Task 2 in Figure 4.4 for example, should be partitioned into 2 subtasks, one of length 3 and one of length 2. This strategy of partitioning long tasks should be preferred because it can enhance the performance of the assignment procedure. Each of the subtasks of length τ_u will occupy entirely one processor. Then, fewer processors will be left for the rest of the tasks. As explained in the next section, the heuristic assignment of a number of tasks to a number of processors performs better for a smaller number of processors.

One can recognize that the longest row corresponds to the longest path of the graph. Heuristics that are based on preventing the solution speed from exceeding the longest path computation may face the problem that the minimum solution time specified by the longest path exceeds the deadline τ_u. The partition of long rows can be considered equivalent to shortening the longest path of the graph. The graph would become shorter and wider resulting in more parallelism.

No special treatment is required for multi-rate algorithms. The problem shown in Figure 4.4 may appear to any row of any solution case. Therefore, this additional partitioning of long rows should be applied on the entire ODE system.

Although the partition of long tasks guarantees that the speed constrain will not be violated, it may still occur that one or more of the tasks is far longer than the other tasks. The appropriate selection of the number of processors would rarely allow this to happen. Nevertheless, the partitioning procedure should provide means to cover this rare case. Consider, for example, a situation in which most of the computation is concentrated in one row of a 10 ODE system $A(x,t)\dot{x}(t)=b(x,t)$. A reason can be that a large number of function generations are required for the evaluation of the elements of that row, while the rest of the system is sparse with relatively simple non-zero elements. In that event, the optimal assignment may result in the situation illustrated in Figure 4.5, resulting in a substantial unbalance.

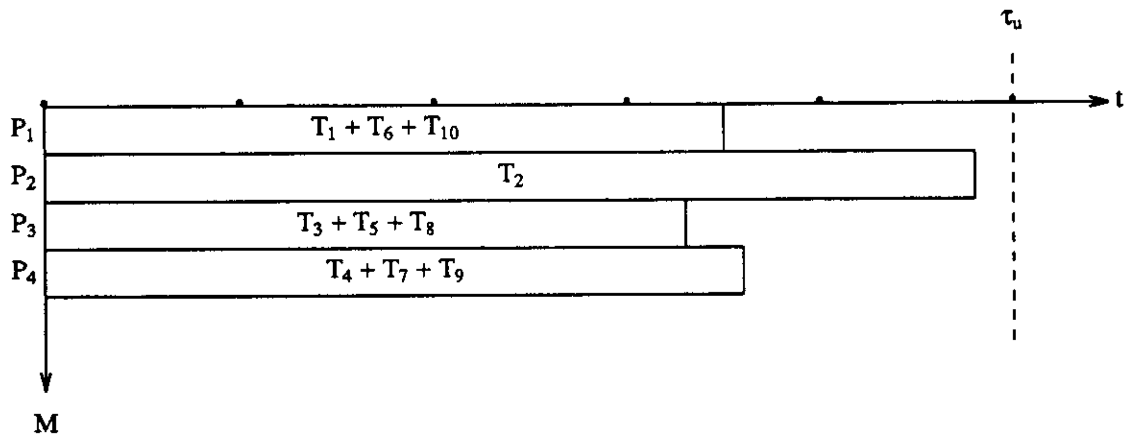


Figure 4.5: Very unequal length tasks may lead to poorly balanced distribution

Then, it is feasible again to partition the long tasks into a number of subtasks. It is explained in the next section, that a heuristic assignment performs better for equal length tasks. It is preferable therefore, that the subtasks be as equal in length as possible. If a long task, say T_i, is of length $L\tau_1+\tau_2$, where $\tau_1 = \frac{1}{M}[(\sum_{j=1}^{N} T_j)-T_i]$, $\tau_2<\tau_1$ and L is an integer, then the task should be partitioned into L+1 tasks of as equal a length as possible. In addition, the number of subtasks should never exceed the number of processors. In case L+1 > M, the long task should be partitioned into M tasks only. Then the M equal length tasks will be assigned to the M processors in a very balanced way, and the assignment of the rest of the tasks on the top of these M tasks will result in an overall balanced distribution. This second partition of long tasks again corresponds to a shortening and widening of the graph resulting in an increase of the parallelism.

For a multi-rate algorithm, this second partitioning has to be performed separately on each of the solution cases. This is because there may be differences between the graphs of the cases such as the problem shown in Figure 4.5. One should apply this partitioning first to the case with the least ODEs and then to the cases in the order of increasing number of ODEs. Then the long tasks of the first case are partitioned first, followed by the partitioning of the long tasks of the second case and so on. This order is justified by the facts that:

1. Cases with fewer ODEs are to be solved more frequently and should therefore have priority.

2. Each case contains the ODEs of the previous cases that would have already been "smoothed".

The entire procedure, for the partitioning of the graphs of multi-rate algorithms used to solve ODE systems in the form $A(x,t)\dot{x}=b(x,u,t)$, is shown in Figure 4.6. The selection of the number of processors is discussed in Section 4.6. For the sake of simplicity, the partitioning of long tasks should be performed as an augmentation of the ODE system by adding new L+1 state variables, rather than partitioning the corresponding column of the graph in Figure 4.1. Such partitioning preserves the simple concept of the partitioning of the graph by rows. Figure 4.7 shows an example of such an augmentation of an ODE system. In this example, the task that corresponds to the *k*th row is decided to be partitioned into 4 subtasks labeled by i, ii, iii and iv. The partitioning is implemented by the insertion of three new rows and three new columns to the system, corresponding to three new state variables x_{N+1}, x_{N+2} and x_{N+3}. Figure 4.7a shows the original ODE system, and 4.7b shows the augmented one.

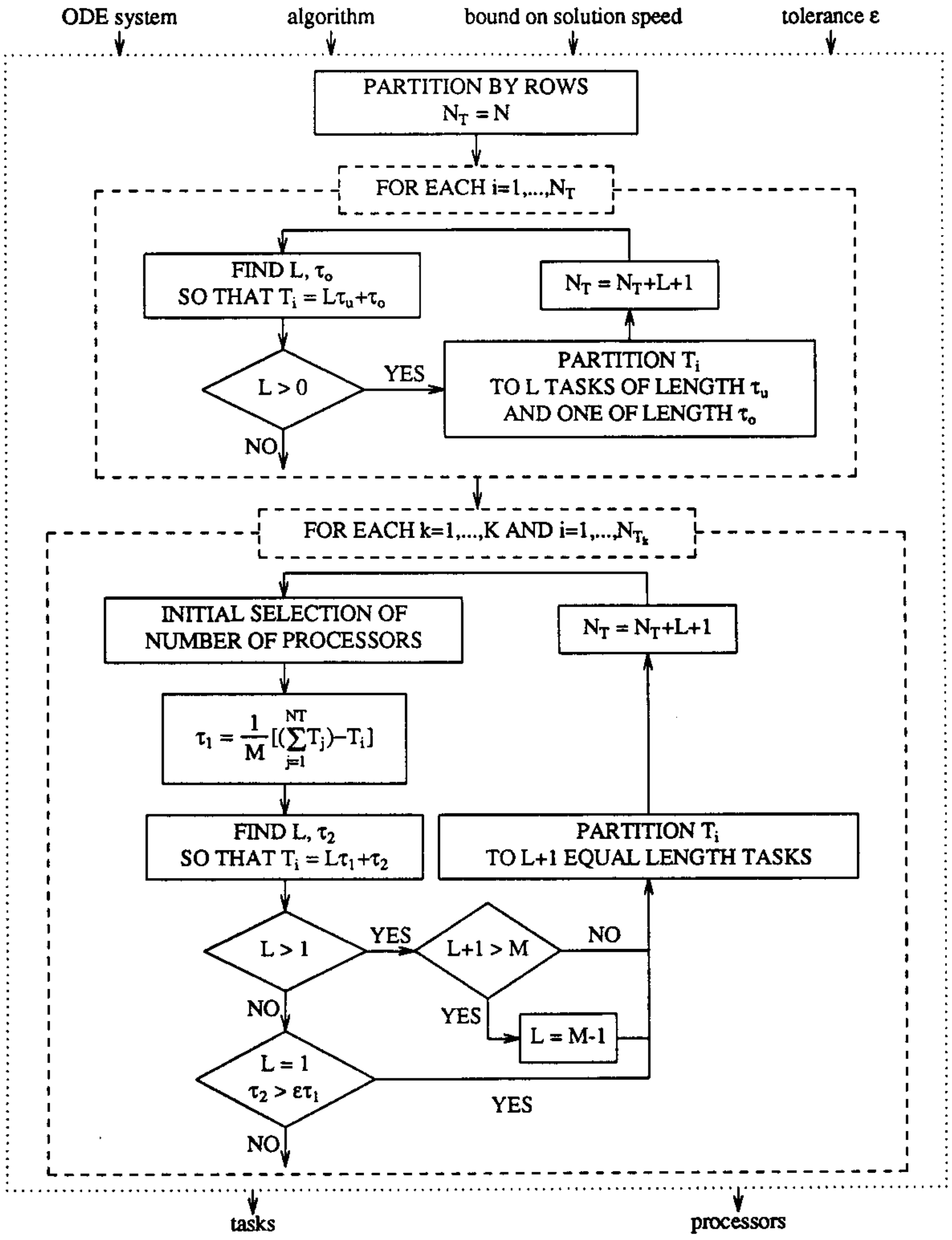

ODE system
algorithm
bound on solution speed
tolerance ε
PARTITION BY ROWS
$N_T = N$
FOR EACH $i=1,\ldots,N_T$
FIND L, τ_o
SO THAT $T_i = L\tau_u + \tau_o$
$N_T = N_T + L + 1$
$L > 0$
YES
PARTITION T_i
TO L TASKS OF LENGTH τ_u
AND ONE OF LENGTH τ_o
NO
FOR EACH $k=1,\ldots,K$ AND $i=1,\ldots,N_{T_k}$
INITIAL SELECTION OF NUMBER OF PROCESSORS
$N_T = N_T + L + 1$
$\tau_1 = \frac{1}{M}[(\sum_{j=1}^{NT} T_j) - T_i]$
FIND L, τ_2
SO THAT $T_i = L\tau_1 + \tau_2$
PARTITION T_i
TO L+1 EQUAL LENGTH TASKS
$L > 1$
YES
$L+1 > M$
NO
NO
YES
$L = M-1$
$L = 1$
$\tau_2 > \varepsilon\tau_1$
YES
NO
tasks
processors

Figure 4.6: Implementation of the partitioning module

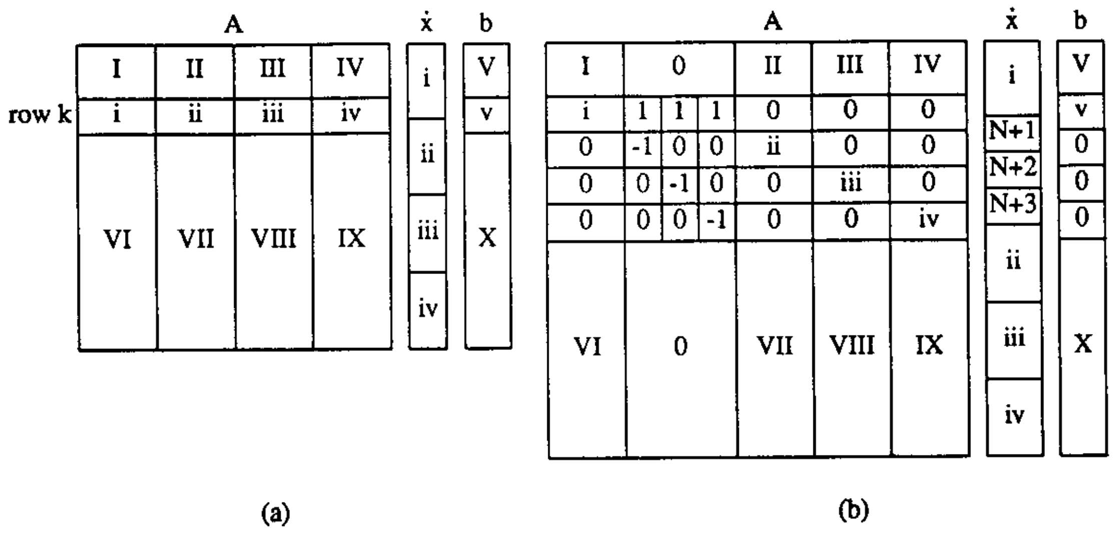


Figure 4.7: Partition of the long task *k* into four subtasks

The suggested partitioning procedure is very simple and easy to implement. It can be considered as an extension of the segmentation technique reported in [Fran78]. It can also be considered as the partitioning part of a "group scheduling" procedure, where entire groups of tasks are suggested to be assigned to processors. The concept of group scheduling is discussed in [Jone79].

Resulting in few loosely coupled tasks, as well as in tasks that are not of very different lengths and guaranteeing that the speed bound will not be violated, it enhances the performance of the assigning and sequencing procedures too. Serving to minimize interprocessor communication, it minimizes the local memory required in each processor as well. This is because a partitioning resulting in more interprocessor communication, would require a larger amount of data to be transferred and equivalently would require more storage in the processors.

The partitioning procedure constitutes a compromise between optimality and simplicity. Greater processing speed may be possible to achieve by clustering the nodes of a more detailed graph so that elementary arithmetic operations are individually allocated. Such a partitioning would however increase the cost of the entire multiprocessor implementation.

4.4 The assigning module

Clustering the nodes of the graph as suggested by the partitioning procedure results in the problem of assigning N_T loosely coupled tasks to a given number of processors. Each of the tasks represents the computation of one of the clusters of nodes. Initially, the number of processors is the one suggested by the INITIAL SELECTION OF NUMBER OF PROCESSORS module in Figure 4.6.

In order to achieve high speed in the solution of the ODE system, it is necessary to determine an assignment that results in:

1. Balanced distribution of the tasks among the processors.
2. Minimization of the idle time of the processor due to interprocessor communication.

In general, the problems of optimal assignment of N_T coupled or uncoupled tasks to M processors under several or no restrictions are NP-complete. In fact, the NP-completeness of the entire scheduling problem is due to the NP-completeness of the assigning problem. High performance heuristics have been suggested for computations of coupled tasks that can be represented in a precedence graph, such as the one in Figure 4.1. The couplings between the tasks in our case are crossprecedences. Each task depends on all the other and all the tasks have to be executed in parallel. Therefore, the computation cannot be represented by a precedence graph. The development of an accurate heuristic for such computation is very difficult. Stochastic scheduling approaches have been suggested [Coff73, Robi79]. Apparently they are inherently not accurate. In addition, the complexity of the queueing modeling of the computation that is involved is relatively high. A much more simple approach is suggested in this section. This approach is based on the assumption that the processors idle time effect of the coupling between the tasks is negligible. Then the tasks can be considered as independent and one can use one of the several widely accepted high performance low complexity heuristics.

Neglecting the effect of the coupling between the tasks is a reasonable assumption because, as can be noticed from Figure 4.1, the impact of the interprocessor communication on the total execution time is relatively very small. Interprocessor communication is already significantly reduced by the partitioning strategy. The optimal sequencing of the execution of the tasks reduces the idle time of the processors even more. In a typical case, the evaluation of the elements of a row of the ODE system would take several thousands of instructions, while the transfer of a datum, such as the value of the element of the vector z, would take only few tens of instructions. As can be noticed from Figure 4.1, this would result in an interprocessor communication time that is of the order of 1% the entire solution time. Having architectural features that allow overlapping of the I/O operations with the arithmetic operations, as shown in Figures 1.2 and 1.3, can reduce the idle time of the processors below 1%. A further reduction can be achieved by combining task preemption with task sequencing.

The criteria to select a heuristic that optimizes the assignment of N_T independent tasks to M processors are:

1. Balance of computational load distribution.

2. Simplicity of the heuristic.

Table 4.1 compares the two heuristics that are at present considered to have the highest performance. The parameter r is the number of tasks assigned to the processor which finishes last. It has been selected to be close to N_T/M as expected for most of the cases.

Table 4.1: Comparison of the LPT and IC heuristics

Parameters	Criterion	LPT	IC
N_T M r	computational complexity	$O(N_T \log_2 N_T + N_T \log_2 M + N_T)$	$O(N_T \log_2 M + N_T)$
	worst performance	$\frac{M-1}{rM}$	$\frac{M-1}{rM-M+1}$
$N_T=100$ M=4 r=20	computational complexity	O(965)	O(300)
	worst performance	3.75%	3.89%
$N_T=100$ M=8 r=10	computational complexity	O(1065)	O(400)
	worst performance	8.75%	9.59%

The LPT (Longest Processing Time first) heuristic [Coff75, Grah69, Coff76] is the closest to optimal, while the IC (1/0-Interchange) heuristic [Finn79] is nearly as optimal but relatively faster. The inferiority of LPT heuristic in terms of computational complexity is due to the fact that it requires initially the sorting of the N_T tasks. The fastest sorting is known to require $O(N_T \log_2 N_T)$ steps. Therefore, the sorting dominates the computational complexity of the LPT heuristic resulting in an inferiority against the IC heuristic. Although the IC heuristic would be the clear choice, in our case the LPT heuristic is preferable. This is because the LPT heuristic can be relieved from the need to sort the tasks before assignment. As explained in Section 4.6, the sorting of the tasks is already performed by the procedure that is suggested to select the number of processors. Figure 4.8 shows the LPT heuristic. Figure 4.9 shows the assignment that the LPT heuristic would suggest for the case of 10 preordered tasks.

For multi-rate algorithms, an assignment has to be suggested for the tasks of each solution case. In all the cases, the number of processors would be the one suggested by the module INITIAL SELECTION OF NUMBER OF PROCESSSORS. The following two alternatives can be considered:

1. Individually optimal assignments for the different cases.

2. Global assignment for all the cases.

Changing the assignment of the tasks over the entire solution at the points of transition from one case to another, would cause an overhead due to the large amount of data required to be transferred between the processors. In addition, such a multi-assignment process would be more complicated to implement. It is therefore desirable to determine a global assignment that is to be used in all the solution cases. A global assignment can be determined by the optimal combination of the assignments that the LPT heuristic suggests for the different cases. Figure 4.10 shows a heuristic

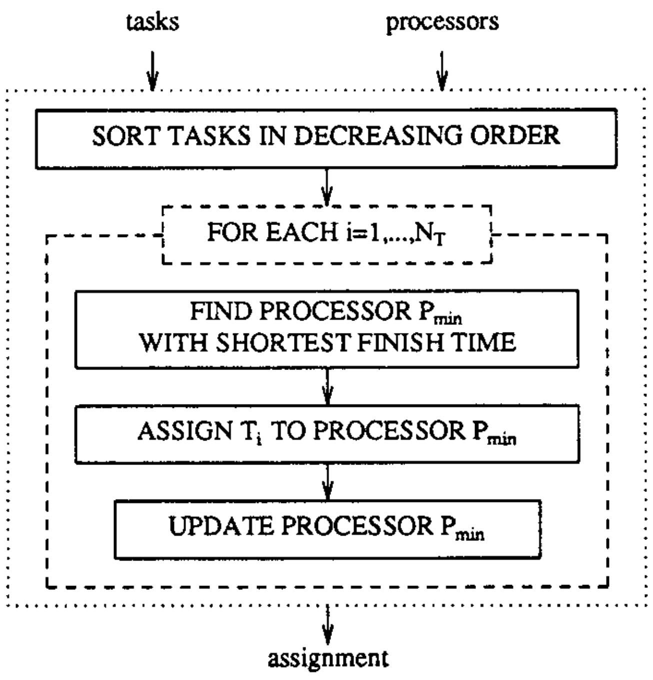

tasks
processors
SORT TASKS IN DECREASING ORDER
FOR EACH i=1,...,N_T
FIND PROCESSOR P_min WITH SHORTEST FINISH TIME
ASSIGN T_i TO PROCESSOR P_min
UPDATE PROCESSOR P_min
assignment

Figure 4.8: The LPT heuristic

way to determine a global assignment. This heuristic constitutes a compromise between optimality and simplicity.

This implementation of the assignment module is very simple. It can be considered as an extension of the simple LPT heuristic for multirate algorithms without the need initially to sort the tasks. In addition, its computational complexity is only of the order of $O(KN_T\log_2 M)$. It also provides an assignment that is fairly close to the optimal, as can be seen from Table 4.1.

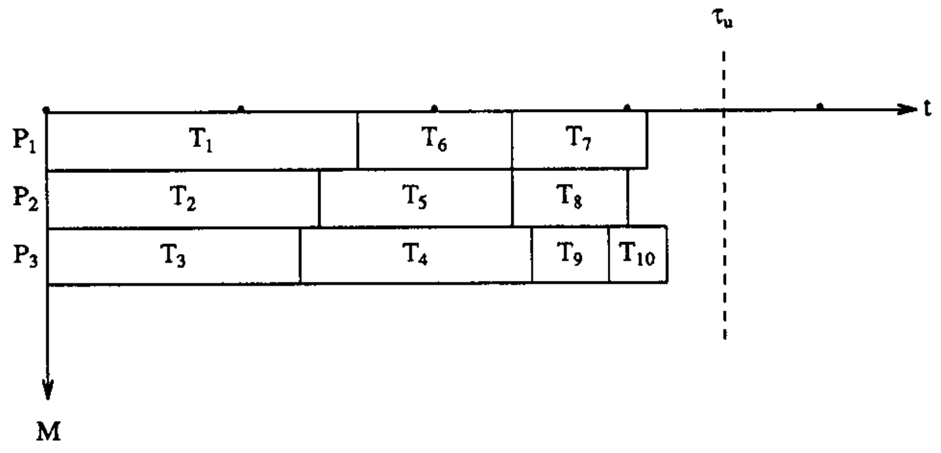


Figure 4.9: An example of LPT assignment

4.5 The sequencing module

The sequencing module determines the priority of execution of the different parts of the computation that is involved in the tasks that are assigned to each processor. The sequencing can be considered at several levels of complexity. At one extreme, one can consider the entire computation as one task. In this trivial case there is no need for sequencing. At the other extreme, one can schedule the execution of the individual elementary operations. This corresponds to the highest level of complexity that apparently can produce a more optimal sequence. Sequencing the execution of the N_T tasks constitutes a compromise between optimality and simplicity.

The contribution of an optimal sequencing to the multiprocessor implementation procedure is the minimization of the processor idle times. To minimize the idle time of a processor, each of the tasks that are assigned to it should be assigned an

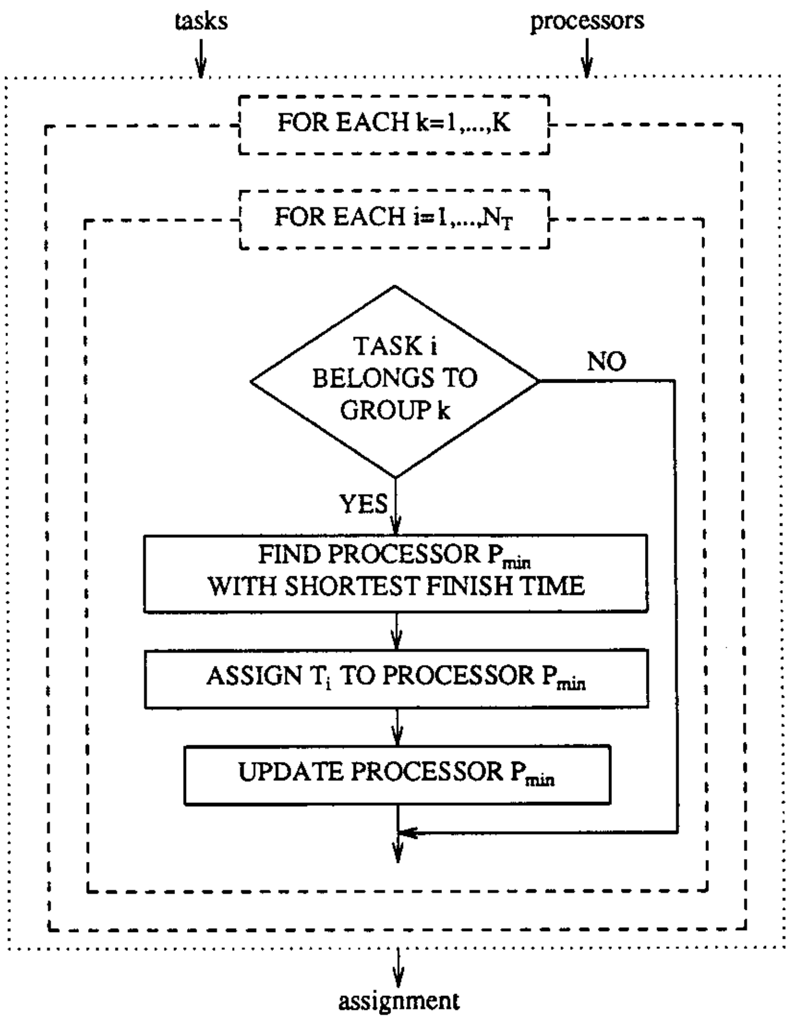

tasks
processors
FOR EACH k=1,...,K
FOR EACH i=1,...,N_T
TASK i BELONGS TO GROUP k
NO
YES
FIND PROCESSOR P_min WITH SHORTEST FINISH TIME
ASSIGN T_i TO PROCESSOR P_min
UPDATE PROCESSOR P_min
assignment

Figure 4.10: The assigning module

execution priority in the order of the amount of its total dependency on the tasks that have been assigned to the other processors. Such a dependency of a task i that has been assigned to the processor j can be measured by the $M \times N_T$ matrix V, defined as:

$$V = (1-E)U \tag{4.2}$$

where the $M \times N_T$ matrix E specifies the assignment of the tasks to the processors and is defined as:

$$E_{ji} = \begin{cases} 1 & \text{if task i is assigned to processor j} \\ 0 & \text{else} \end{cases} \tag{4.3}$$

and the $N_T \times N_T$ matrix U represents the mutual coupling between the tasks and is defined as:

$$U_{ij} = \begin{cases} 1 & \text{if row i depends on } x_j \\ 0 & \text{else} \end{cases} \tag{4.4}$$

Sequencing smaller tasks, such as individual elements of the rows of the ODE system or individual elementary operations, can be based on the expression (4.2) with a definition of the matrices e and U similar to (4.3) and (4.4).

In case of multi-rate algorithms, the tasks that are assigned to processors differ for every solution case. Therefore, a separate scheduling has to be performed for each case as shown in Figure 4.11.

4.6 The module for selection of the number of processors

The optimization of the number of processors contributes to:

1. Minimization of the hardware cost.

2. Maximization of the solution speed.

In particular, the optimization of the number of processors M requires the determination of the minimum number of processors M_o, that the multiprocessor computer system should utilize to achieve the desirable speed S_u in the solution of the ODE system.

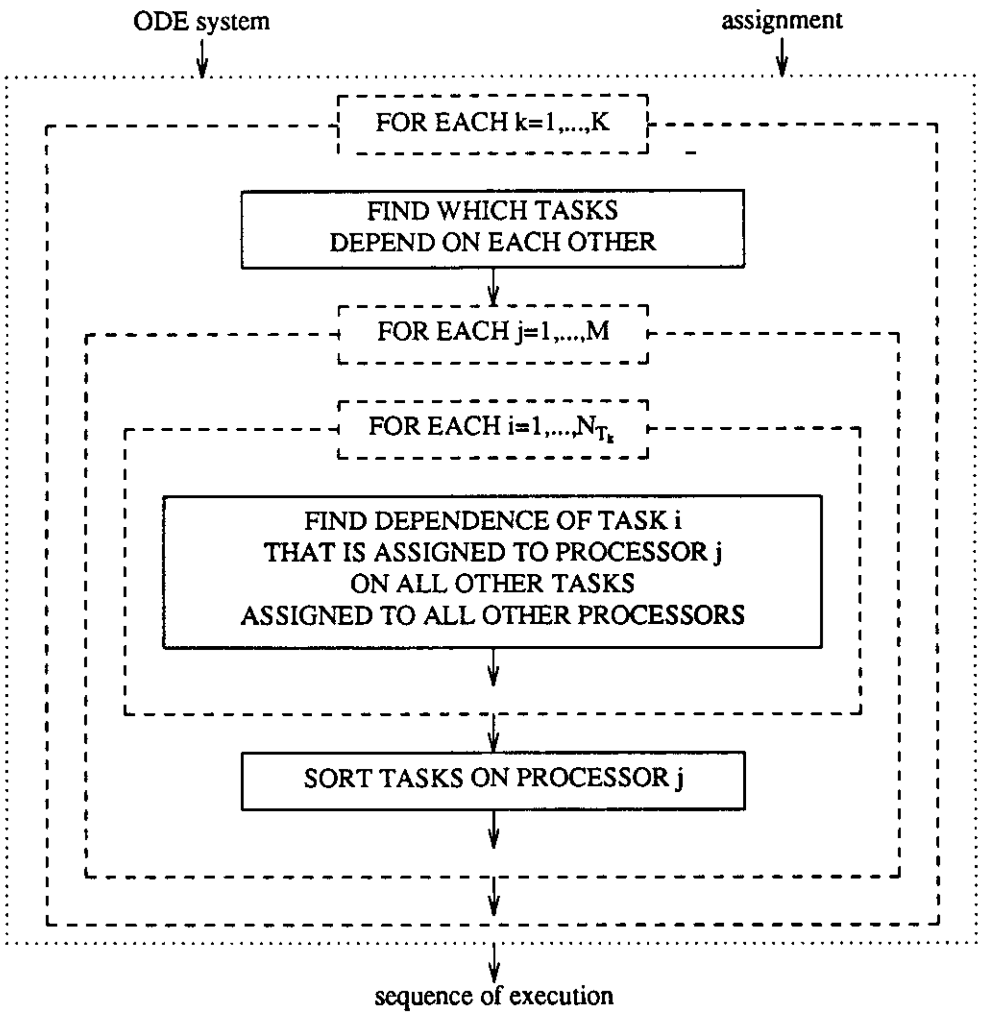

ODE system
assignment
FOR EACH k=1,...,K
FIND WHICH TASKS DEPEND ON EACH OTHER
FOR EACH j=1,...,M
FOR EACH i=1,...,N_{T_k}
FIND DEPENDENCE OF TASK i THAT IS ASSIGNED TO PROCESSOR j ON ALL OTHER TASKS ASSIGNED TO ALL OTHER PROCESSORS
SORT TASKS ON PROCESSOR j
sequence of execution

Figure 4.11: The sequencing module

The solution time τ depends monotonically on the number of processors, as shown in Figure 4.12. Notice that, as the partitioning procedure, shown in Figure 4.6, suggests that for each solution case, say #k, N_{T_k} tasks are to be distributed to the M processors, it is necessary that:

$$1 \leq M_o \leq N_{T_k} \tag{4.5}$$

for each iteration k=1,...,K of the second loop in Figure 4.6. M_o can be determined to be the maximum integer M that satisfies the relation $\tau(M) \leq \tau_u$. M_o could be derived from such a relation if $\tau(M)$ could be expressed analytically. However, the formulation of such an analytic expression is extremely difficult.

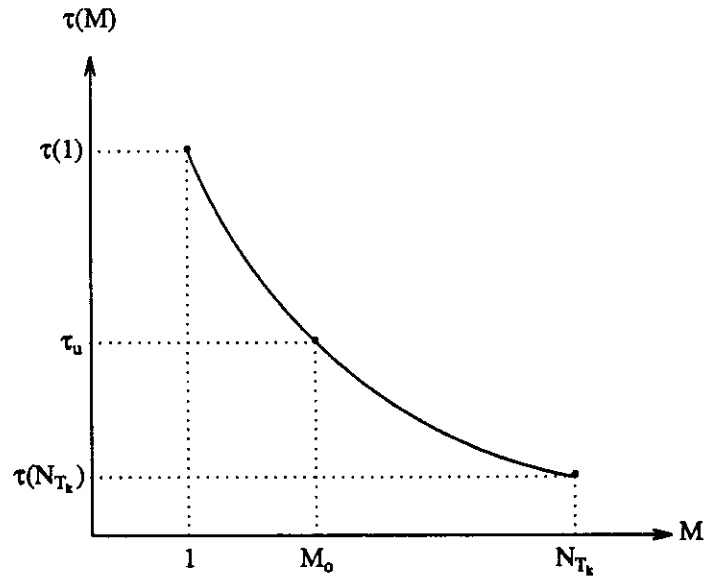


Figure 4.12: The solution time decreases with the number of processors

Alternatively, the determination of M_o could also be seen as a combinatorial problem, such as the assigning problem. This is the problem of determining the minimum number of processors that can execute a graph of N_{T_k} loosely coupled tasks in a time that does not exceed a predefined deadline τ_u. Such combinatorial problems are again NP-complete. A heuristic has to be selected to provide a quick approximate solution. The heuristics that have been proposed for coupled tasks, such as in [Fern73], apply again for precedence type of coupling. They cannot be used in our case of cross-precedence couplings. Similarly the approach to determine an optimal assignment, the N_{T_k} tasks for the case #k can be considered as independent. This assumption introduces again a compromise between optimality and simplicity. The problem of determining the minimum number of processors that can execute a set of independent tasks with a predefined deadline, can be modeled as a classical bin-packing problem. The FFD (First Fit Decreasing) heuristic [John74, Coff78], shown in Figure 4.13, has been used successfully to find a quick approximate solution to the bin-packing problem. It suggests a number of processors $\hat{M}_o$ which in the worst case is bounded, relative to the optimal number M_o, as:

$$\hat{M}_o \leq \frac{11}{9} M_o + 4 \tag{4.6}$$

In addition, its computational complexity is only of the order of $O(N_{T_k} \log_2 N_{T_k})$. Therefore, it combines high performance with simplicity and low computational complexity. Figure 4.14 shows the number of processors that the FFD heuristic would suggest for the case of the 10 preordered tasks shown in Figure 4.9.

4.7 The performance improvement module

The heuristics, that are proposed in Sections 4.3 through 4.6 to determine the optimal multiprocessor implementation of algorithms, are claimed to be of high performance. They are expected in most cases to provide a multiprocessor implementation that leads to maximum speed of the algorithms. In some cases however, due to the nonoptimality of the heuristics or due to the assumptions that are introduced, the multiprocessor implementation that is suggested by the heuristics may not be as

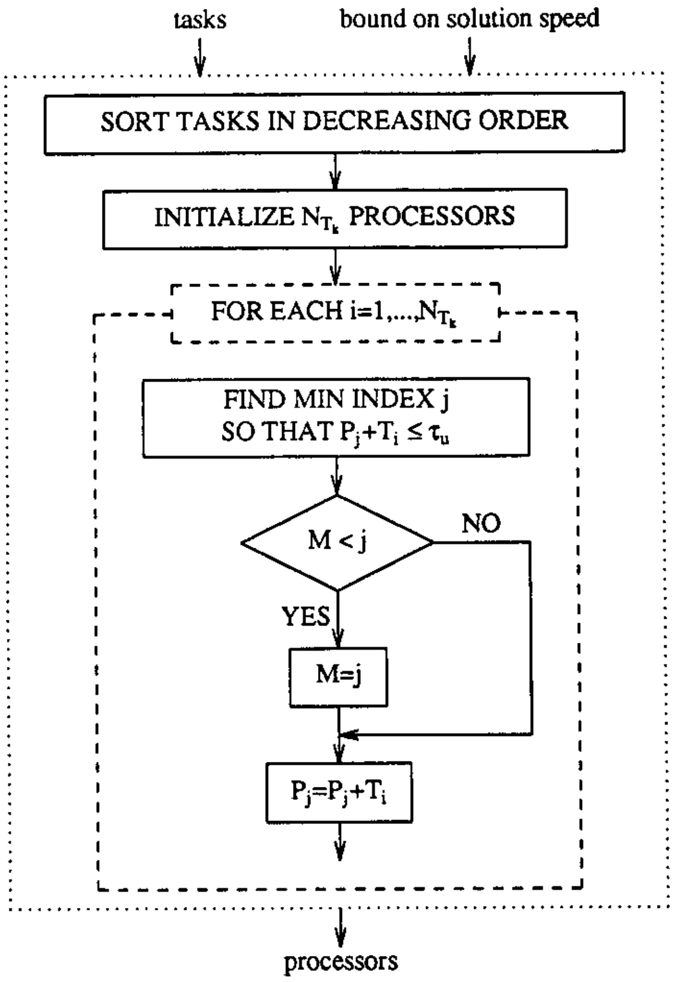

tasks
bound on solution speed
SORT TASKS IN DECREASING ORDER
INITIALIZE N_{T_k} PROCESSORS
FOR EACH $i=1,...,N_{T_k}$
FIND MIN INDEX j SO THAT $P_j+T_i \leq \tau_u$
$M < j$
NO
YES
$M=j$
$P_j=P_j+T_i$
processors

Figure 4.13: The module for the selection of the number of processors

close to the optimum as one might require. It may also happen that the speed of the algorithms finally falls below the desirable lowest bound, although the heuristics attempt to prevent such a situation. Or it may happen that the estimation of the number of processors by the heuristics was too conservative, that is, fewer processors would suffice to achieve the desirable speed. As discussed in Section 4.2, the heuristics should be enhanced with capabilities to asymptotically improve their

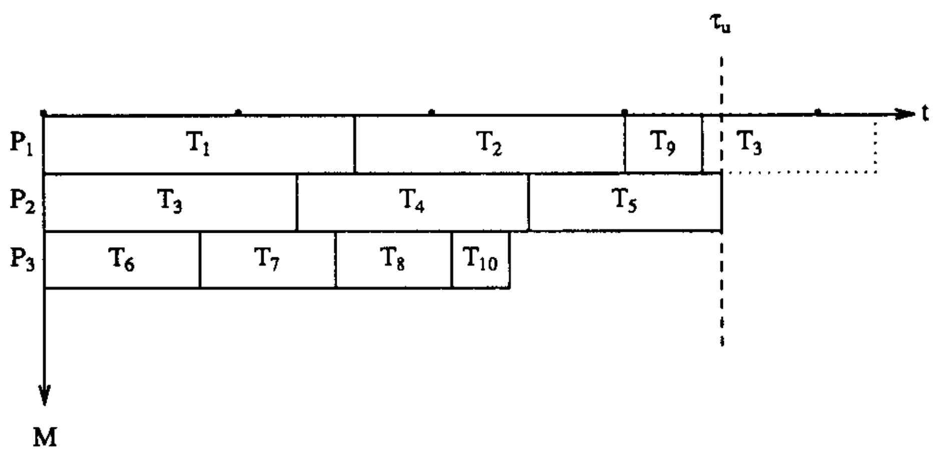


Figure 4.14: An example of FFD determination of the number of processors

results. The entire procedure of the multiprocessor implementation consists of a combination of heuristics and improvements features.

The detection of the need for improvement can be based on an estimation of the speed of the solution of the ODE system. Any of the commercially available discrete event simulators or simulation languages can provide such an estimate. The multiprocessor implementation procedure should include a module that generates automatically the code of such a simulator. This code should include:

1. Emulation of hardware: A representation of the architecture of the multiprocessor computer.
2. Emulation of software: A representation of the computation to be executed by the computer.

The criteria to select a simulator include:

1. The simulator should have enough features that allow the representation of all the important hardware and software parts of the computation.

2. The simulator should be simple enough so that the generation of its code is an easy task.

3. The simulator should be fast enough so that it does not slow down the entire procedure.

The main reasons for failing to achieve a desirable solution speed are related to the compromise between maximum solution speed and simplicity and speed of the procedure. These reasons can be:

1. The decision to partition the computation into few large tasks.

2. The decision to neglect the coupling between the tasks.

The first decision is the most important feature of the procedure that serves the criterion of having a fast procedure. No improvement of the results of the heuristics should be sought through the consideration of smaller tasks. The second decision serves the important criterion of simplicity of the procedure. Although again such a decision should not be dropped, additional care can be taken to reduce its effects.

The coupling between the tasks may result in processors idle times and slow down the algorithms further than the heuristics can predict and control. Processor idle times could be controlled by controlling the interprocessor communication that is required. Forcing highly coupled tasks to be executed by the same processor may reduce the interprocessor communication required. Therefore, an improvement of the multiprocessor implementation can be attempted by iteratively merging the pair

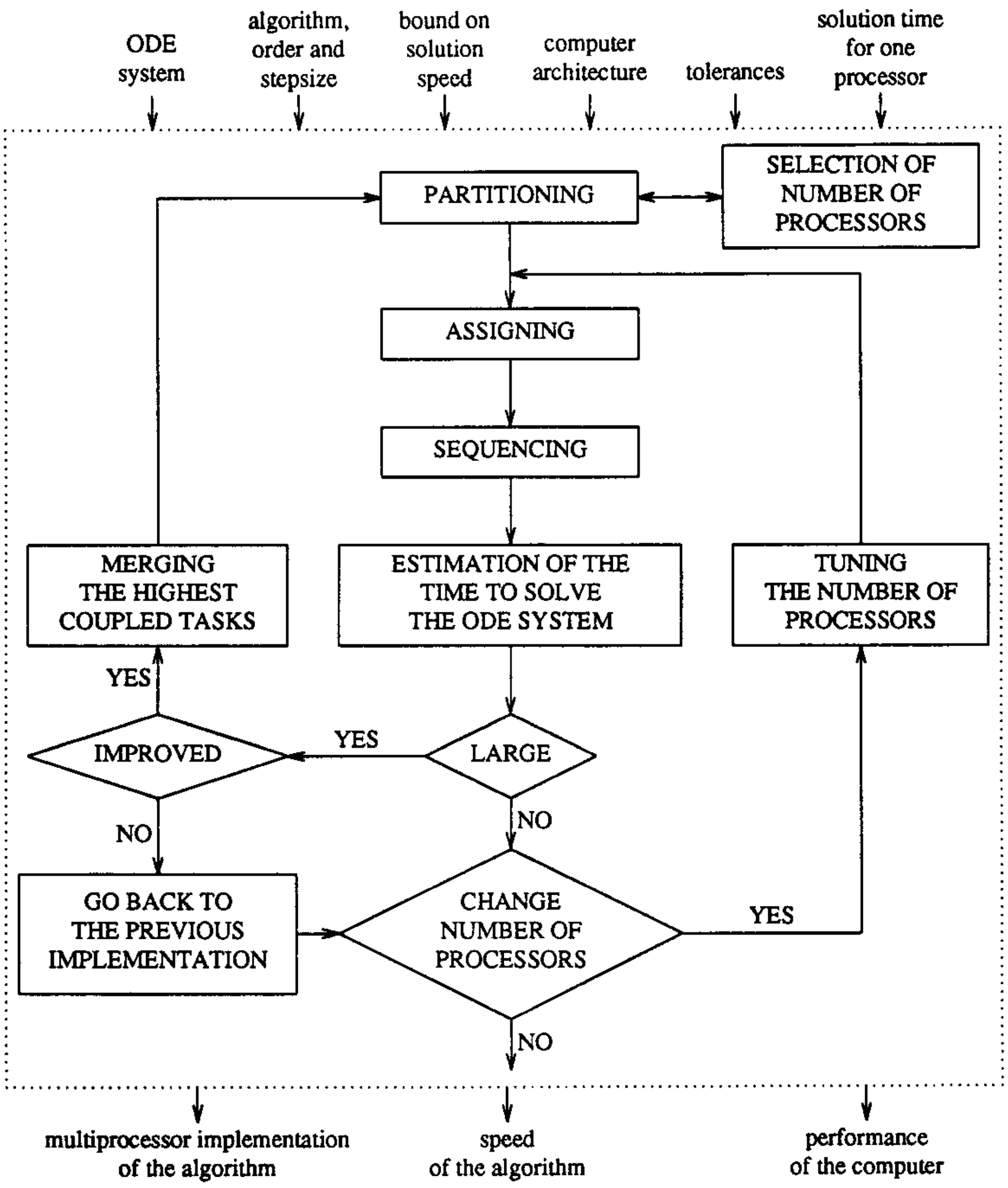

ODE system
algorithm, order and stepsize
bound on solution speed
computer architecture
tolerances
solution time for one processor
PARTITIONING
SELECTION OF NUMBER OF PROCESSORS
ASSIGNING
SEQUENCING
MERGING THE HIGHEST COUPLED TASKS
ESTIMATION OF THE TIME TO SOLVE THE ODE SYSTEM
TUNING THE NUMBER OF PROCESSORS
YES
IMPROVED
YES
LARGE
NO
NO
GO BACK TO THE PREVIOUS IMPLEMENTATION
CHANGE NUMBER OF PROCESSORS
YES
NO
multiprocessor implementation of the algorithm
speed of the algorithm
performance of the computer

Figure 4.15: The multiprocessor implementation procedure

THE TIME TO SOLVE THE ODE SYSTEM, while Figure 4.17 shows an implementation of the module MERGING HIGHEST COUPLED TASKS.

A heuristic iterative tuning of the number of processors can be based on Figure 4.12. In particular, as discussed in Section 4.6, M_o can be determined to be the largest integer M that satisfies the relation $\tau(M) \le \tau_u$. A piecewise linear approximation between already available values can be used to express $\tau(M)$. Then, at each improvement iteration a new value of V_o can be determined by linear interpolation, as shown in Figure 4.18. This iterative improvement requires initially the availability of at least two values of $\tau(M)$ for two different number of processors. These two values can be:

1. The solution time for the number of processors that the module SELECTION OF THE NUMBER OF PROCESSORS suggests.

2. The solution time for one processor that can easily be estimated with an actual run of the simulation without any need for multiprocessor implementation of the algorithm.

It is easy to show that at every iteration step the new value of M, say k+1, can be computed from:

$$M_{k+1} = M_k \frac{\tau(M_{k-1})-\tau_u}{\tau(M_{k-1})-\tau(M_k)} \pm M_{k-1} \frac{\tau_u-\tau(M_k)}{\tau(M_{k-1})-\tau(M_k)}, \quad k \ge 2 \tag{4.7}$$

The "+" sign corresponds to interpolation between $\tau(M_{k-1})$ and $\tau(M_k)$, while the "-" sign corresponds to extrapolation. The module for TUNING NUMBER OF PROCESSORS, shown in Figure 4.19, includes a capability to detect if an interpolation or an extrapolation is needed.

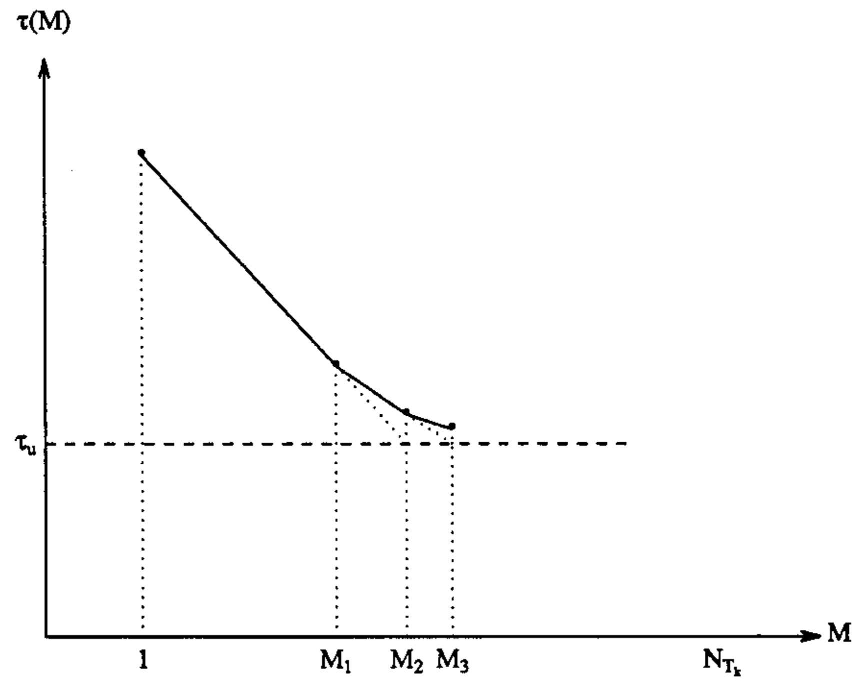


Figure 4.18: The iterative tuning of the number of processors

The implementation of the entire module of performance improvement that is proposed in this section, is fairly simple to implement. It is expected to require no more than a few iterations. It does not therefore, significantly increase the complexity and decrease the speed of the multiprocessor implementation procedure. In addition, it allows the asymptotic approximation of the maximum solution speed by tuning appropriately the tolerances of the decisions involved.

It should be recognized that the multiprocessor implementation procedure offers for free a prediction of the performance of the computer to be used to solve the

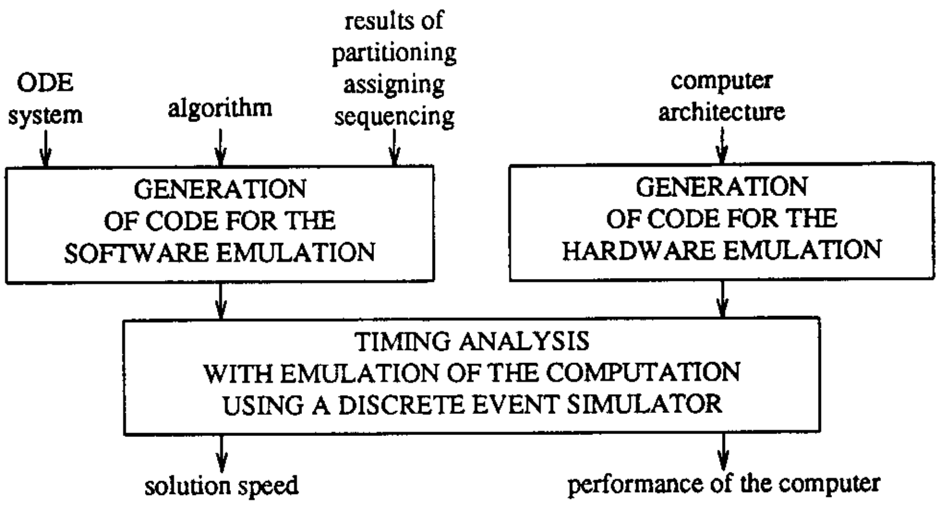

results of partitioning assigning sequencing
ODE system
algorithm
computer architecture
GENERATION OF CODE FOR THE SOFTWARE EMULATION
GENERATION OF CODE FOR THE HARDWARE EMULATION
TIMING ANALYSIS WITH EMULATION OF THE COMPUTATION USING A DISCRETE EVENT SIMULATOR
solution speed
performance of the computer

Figure 4.16: The module for estimating the solution time

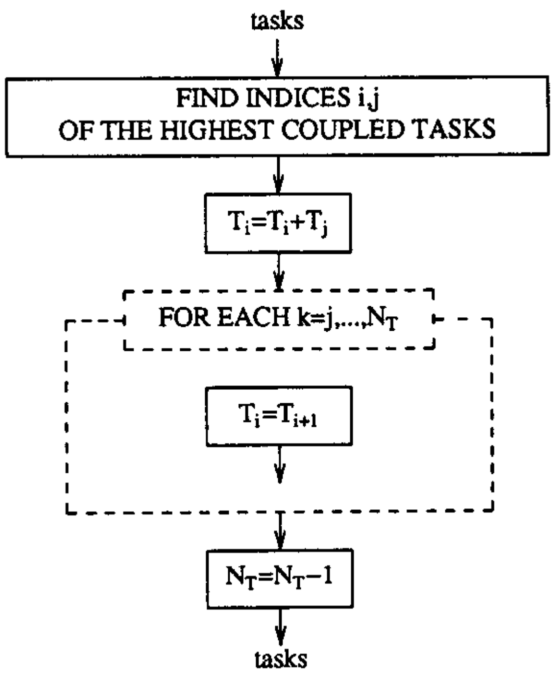


Figure 4.17: The module for merging the highest coupled tasks

the interprocessor communication operations with the arithmetic operations. However, the procedure is fairly general to apply to different multiprocessor architectures, different algorithms and different formulations of the ODE systems.

The procedure is developed to meet the criteria of simplicity and of maximum speed of the algorithms. The first criterion can be achieved by developing a fast and easy-to-implement procedure. The second criterion can be achieved by a balanced distribution of the computational load and a minimization of the idle time of the processors. The problem of the determination of the optimal multiprocessor implementation is an NP-complete problem. A heuristic approach is suggested to provide a quick approximate solution. Figure 4.3 shows the necessary components of the procedure to meet this criterion.

The module of partitioning requires the partitioning of the computational graph that represents the computation that is involved, as shown in Figures 4.1 and 4.2, into few large loosely coupled clusters of nodes, called tasks. The approach that is proposed is shown in Figure 4.6. It is based on the idea of partitioning the ODE system by rows. The set of tasks that are to be produced, is expected to guarantee that the speed of the multiprocessor algorithm will not exceed a predefined lower bound. In addition, it enhances the performance of the assigning and sequencing processes.

The module of selecting the number of processors requires the determination of the minimum number of processors that can solve the ODE system with a speed that does not exceed the lower bound. Figure 4.13 shows a possible approach that is an extension of the FFD heuristic, which attempts to provide a quick approximate solution to the classical bin-packing problem. This approach is based on neglecting the coupling between the tasks.

The module of assigning requires the assignment of the tasks to the processors for execution. The approach suggested, as shown in Figure 4.10, is based again on neglecting the coupling between the tasks, and is an extension of the LPT heuristic, which provides a quick approximate solution to the problem of the assignment of independent tasks.

The module of sequencing requires the assignment of execution priority to the tasks that are to be executed on each processor. The priorities are suggested to be assigned according to the dependence of each task on the tasks that are to be executed in all the other processors, as shown in Figure 4.11.

The module of performance improvement suggests ways to improve the results of the previous four modules. It allows an infinite approximation of the maximum speed of solution of the ODE system through iteration loops, as shown in Figure 4.15. The detection of the need for improvement is suggested to be performed using any of the commercially available discrete event simulators as shown in Figure 4.16. For a particular number of processors, merging the highest coupled tasks into a single task, as shown in Figure 4.17, may reduce the idle time of the processors and increase the solution speed. Further improvement can be achieved by tuning appropriately the number of processors, as shown in Figure 4.19.

5. A BENCHMARK ODE SYSTEM

This chapter is devoted to the selection of an ODE system that can serve as benchmark for the performance analysis of the procedures for algorithm selection, that is introduced in Chapter 2 and 3, and for multiprocessor implementation of algorithms, that is introduced in Chapter 4. The first section discusses the requirements for the characteristics of the benchmark. It is explained that synthetic benchmarks have to be developed. This leads in the next section to a procedure for the development of a synthetic benchmark. A validation of the procedure is presented in the third section. The development of the benchmarks that are to be used for the two performance analyzes are presented in the last two sections.

5.1 Requirements for the characteristics of the benchmark

The criteria for the selection of the benchmark ODE system are:

1. Validity: The benchmark must have all the important characteristics and only those of the ODE systems that appear in the models of the dynamics of space structures. As explained in Section 3.1, these ODE systems are:

    1. Large

    2. Highly nonlinear

    3. Highly oscillating

    4. Lightly damped

    5. In the form $A(x,t)\dot{x}(t)=b(x,u,t)$

In addition, the parameters that achieve these characteristics should be sufficient for the performance analyzes.

2. Simplicity: The benchmark must be as small and simple as possible so that the performance analysis tasks are simplified.

It should be recognized that an ODE system involved in the model of a particular space structure may have only some of the above characteristics. Therefore, the performance analyzes require a set of different benchmark ODE systems representing different space structures so that all together cover all the required characteristics. The task of specifying and constructing such a set of ODE systems however, is very difficult. Ready to use models for numerous different space structures are not available. Even if such models exist, their conversion to the semiexplicit form $A(x,t)\dot{x}=b(x,u,t)$ poses additional problems. It is more practical to develop a synthetic benchmark that includes parameters having the required characteristics. The benchmark can then be tuned for the needs of the specific performance analysis. The synthetic benchmark should meet the above two criteria. Let us specify the characteristics that it must have.

Large

The dimension N of the ODE systems that appear in the mathematical models of the dynamics of modern space transportation systems is usually of the order of one to two hundred. A simplified modeling of a space shuttle, represented by 10 bodies interconnected by 10 hinges, led to 130 ODEs [Gluc85]. Extensive experimentation with benchmarks of that size is excessively time consuming. As a compromise between realism and simplicity, it has been decided to develop ODE systems with dimensions of the order of:

1. $N=12$ for the analysis of the performance of the procedure for selection of algorithms.

2. $N=48$ for the analysis of the procedure for multiprocessor implementation of algorithms.

Highly Nonlinear

The models of the dynamics of space structures include typically terms of the form:

1. η (constant)

2. $\alpha_i x_i$ (linear)

3. $\beta_{ij} x_i x_j$

4. $\gamma_{ij} x_i \cos(x_j)$

5. $\delta_{ijk} x_i x_j \cos(x_k)$

6. $\xi_i \phi(x_i)$, where $\phi(x)$ is a nonlinear function of x that contains discontinuities

The size of the nonlinear terms is irrelevant to the multiprocessor implementation procedure. The algorithm selection procedure however, should be tested for various sizes of the nonlinear terms. In particular, it has been decided to consider the following two cases:

1. The size of the coefficients of the nonlinear terms is of the same order of those of the linear terms.

2. The size of the coefficients of the nonlinear terms is one tenth of the order of those of the linear terms.

Highly Oscillating and Lightly Damped

The typical frequency range of the oscillations is 0.1 - 100 Hz. The typical duration of the transient is 300 secs. These have the impact that the benchmark when linearized should have eigenvalues $\{\lambda_1, \ldots, \lambda_N\}$ with real part up to -0.015 and imaginary part in the range j0.628 - j628.318. It should be recognized that the eigenvalues always appear as complex conjugates pairs.

The linearized benchmark is of the general form:

$$\dot{x} = Fx+Gu \tag{5.1}$$

where the matrix F represents the interaction between the state variables, and the matrix G represents the effect of the inputs on the state variables. The mathematical model consists of real number expressions and this must also be true of the elements of the matrices F and G. In addition, the eigenvalues of the ODE system are the eigenvalues of the matrix F. According to the Gerschgorin theorem [Stew73], the matrix F has eigenvalues that lie in the discs in the complex plane defined as:

$$\left|\lambda_i - F_{ii}\right| \le \sum_{\substack{j=1 \\ j\neq i}}^{N} \left|F_{ij}\right|, \quad i=1,\ldots,N \tag{5.2}$$

and shown in Figure 5.1. For a benchmark to have eigenvalues that are not more than a tolerance ε displaced from a given set of eigenvalues, the matrix F should have:

1. 2×2-block real number diagonal elements that are related to these given

eigenvalues. Every pair of complex conjugates eigenvalues $\lambda_1=\sigma+j\omega$ and $\lambda_2=\sigma-j\omega$ correspond to the block:

$$\begin{bmatrix} \sigma & \omega \\ -\omega & \sigma \end{bmatrix}$$

2. Very small nondiagonal elements so that the sum of each row of F does not exceed ε. The nondiagonal elements represent the coupling between the state variables. This coupling is related to the coupling of the motion of the bodies and therefore must be represented by real numbers.

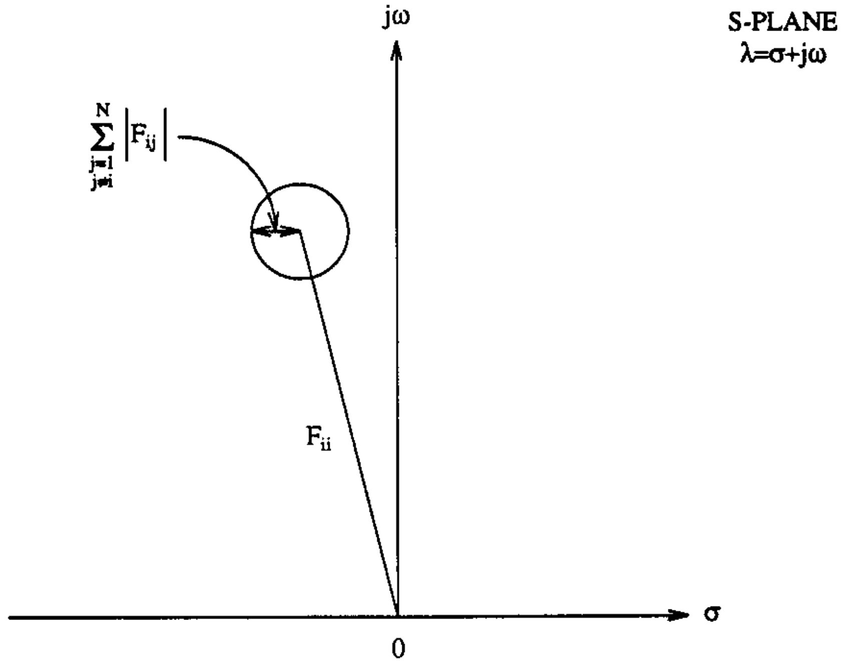


Figure 5.1 : Each eigenvalue λ_i of the matrix F lies within the disc

The duration of the transient and the frequency range are irrelevant to the multiprocessor implementation procedure. The selection of an algorithm however, depends strongly on both. Since the ODE systems that describe the dynamics of different space structures may differ significantly only in their frequency range, it has been decided that the performance of the algorithm procedure is evaluated for:

1. Constant 300 secs transient.

2. Two frequency ranges that correspond to maximum frequencies of 50 and 90 Hz.

The duration of the transient is typical and is selected to be long enough so that the algorithm selection procedure is tested under "worst case" conditions. The two maximum frequencies are also typical and have been selected so that one of them is close to and the other far from 100 Hz. According to (3.1), 100 Hz, as well as 10 Hz or 1 Hz, is a bounding maximum frequency that requires a step size larger by one order of magnitude. It should be expected that the more close to such a bounding maximum frequency the less accurate the prediction of the optimal orders and step sizes.

Form $A(x,t)\dot{x}(t)=b(x,u,t)$

The manipulation of ODE systems with time dependent terms, especially their linearization and eigenvalue analysis, is in general too complicated to be suitable for a benchmark. Neglecting time dependency in the development of the benchmark is a very useful simplification. For the same reason, the separation of the terms dependent on x and the terms dependent on u in the vector b is another useful simplification. Then the benchmark can be formulated as:

$$A(x)\dot{x} = \zeta_1(x)+\zeta_2(u) \tag{5.3}$$

The equilibrium point x_0 of the ODE system is defined as:

$$x_0=\lim_{t\to\infty} x(t) \tag{5.4}$$

At the equilibrium point it is $\dot{x}(t)=0$ or equivalently $b(x_0,u_0)=0$, where:

$$u_0=\lim_{t\to\infty} u(t) \tag{5.5}$$

Then the linearization of the benchmark leads to the following linear ODE system:

$$[A(x)]_{x_0}\dot{x} = [\frac{d\zeta_1(x)}{dx}]_{x_0} x+[\frac{d\zeta_2(u)}{du}]_{u_0} u \tag{5.6}$$

The following equalities can identify the matrix A and the vector b of the benchmark:

$$[A(x)]_{x_0} = L \tag{5.7}$$

$$[\frac{d\zeta_1(x)}{dx}]_{x_0} = LF$$

$$[\frac{d\zeta_2(u)}{du}]_{u_0} = LG$$

An obvious solution of these equations is the following:

$$\zeta_1(x) = LFx \quad \text{and} \quad \zeta_2(u) = LGU \tag{5.8}$$

The equilibrium point x_0 can be determined from the equation $\zeta_1(x)+\zeta_2(u) = 0$ which results in:

$$x_0 = -F^{-1}Gu_0 \tag{5.9}$$

The conclusion of all the above discussion is that the benchmark can be formulated as:

$$A(x)\dot{x}(t) = LFx+LGu \tag{5.10}$$

where the matrices F and G are as described above and:

$$\begin{aligned} L &= [A(x)]_{x_0} \\ x_0 &= -F^{-1}Gu_0 \\ u_0 &= \lim_{t\to\infty} u(t) \end{aligned} \tag{5.11}$$

The elements of the matrix A(x) consist of linear and nonlinear terms such as those discussed above. The sparsity of the matrix and the complexity of its nonzero elements are irrelevant to the selection of the algorithms but they are important factors to the multiprocessor implementation of algorithms. Therefore, it has been decided to analyze the performance of the multiprocessor implementation procedure for various sparsities and complexities of the matrix A(x). The necessary set of sparsities and complexities is selected in Chapter 7.

5.2 Procedure for the development of the benchmark

Summarizing the discussion in the preceding section, the development of a specific benchmark takes the following steps:

1. Selection of an interesting mechanical structure corresponding to a simplified model of a space structure. This structure can be defined by:

    a. The number of bodies nb.

    b. The relative position and the interconnection of the bodies.

c. The relative magnitude of the masses and elasticity of the bodies.

d. The degrees of freedom (d.o.f.) of the bodies $df_1,\ldots,df_{nb}$.

2. Calculation of the dimension of the benchmark ODE system as:

$$N=\sum_{i=1}^{nb} df_i$$

3. Selection of a realistic set of eigenvalues $\{\lambda_1,\ldots,\lambda_N\}$. These eigenvalues must be nearly imaginary and pairwise conjugate. The assignment of their values can be based on the relative magnitude of the masses and the elasticity of the bodies. The smallest and most flexible body corresponds to eigenvalues with the maximum imaginary part.

4. Selection of the tolerance ε to be an upper bound for the difference between the specified eigenvalues and the actual eigenvalues as shown in Figure 5.1.

5. Development of the matrix F so that:

   a. For every pair of conjugate eigenvalues $\lambda_i=\sigma+j\omega$ and $\lambda_j=\sigma-j\omega$ there is a 2×2 block along the diagonal in the form

$$\begin{bmatrix} \sigma & \omega \\ -\omega & \sigma \end{bmatrix}$$

   b. The nondiagonal elements are very small so that the sum of the rows of F does not exceed ε. The nondiagonal elements should represent the coupling between the state variables.

6. Selection of the input vector u defined by the number of inputs, the bodies to which they are applying and their mathematical model. Each input should

include as many components as there are the d.o.f. of the body to which it is applied. Each of these components appears as a nonzero element in the vector u. The rest of the elements are 0.

7. Development of the matrix G with elements that represent the coupling of the inputs with the state variables.

8. Development of the matrix A(x) defined by:

    a. The sparsity of the matrix A. The position of the zero elements of the matrix can be selected randomly with the help of a random number generator for a specified sparsity s%.

    b. The complexity of the nonzero elements of the matrix A that are expressed as:

    $$A_{ij}=\eta_{ij}+\sum_{k}\alpha_{ij_k}x_k+\sum_{k}\sum_{l}\beta_{ij_{kl}}x_k x_l+\sum_{k}\sum_{l}\gamma_{ij_{kl}}x_k\cos(x_l)+\sum_{k}\sum_{l}\sum_{m}\delta_{ij_{klm}}x_k x_l\cos(x_m)+\sum_{k}\xi_{ij_k}\phi(x_k)$$

    $\phi(x)$ represents a nonlinear function of x with discontinuities. For each of the nonzero elements of A there is a need for the selection of the constant η, the N×1 vector α, the N×N matrix β, the N×N matrix γ, the N×N×N matrix δ and the N×1 vector ξ. These vectors and matrices have to be very sparse in order to keep the benchmark ODE systems reasonably simple. The position of the zero elements can be generated randomly. The values of the nonzero elements can be generated randomly as well.

9. Computation of the N×1 vector $u_0 = \lim_{t\to\infty} u(t)$.

10. Computation of the equilibrium point $x_0 = -F^{-1}Gu_0$.

11. Computation of the matrix $L = A(x_0)$, the linearized matrix $A(x)$ around the equilibrium point.

12. Computation of the vector $b(x,u) = LFx+LGu$.

13. Construction of the overall benchmark $A(x)\dot{x} = b(x,u)$

5.3 Validation of the procedure

The ODE systems that the procedure produces are simple and easy to manipulate without omitting any of the important characteristics. They may lack reality only in the following respects:

1. Their size may be smaller than real problems so that the results of the performance analyzes may be a little super-optimistic. However, they are large enough to constitute acceptable benchmarks.

2. They do not include time varying terms. However, neither the procedure for the algorithm selection nor the procedure for the multiprocessor implementation of algorithms depend heavily on the existence of time varying terms in the mathematical models. If needed, damping time varying terms, such as e^{-t}, can be added to the elements of the matrix A or the vector b.

3. The complexity of their nonzero elements may be lower than in real systems. The procedures for the algorithm selection and the multiprocessor implementation however, depend only on the relative, rather than absolute, complexity of the rows or the elements of the rows of the ODE system.

4. The vector b does not include any nonlinearities except those introduced by the input vector u. The matrices $\zeta_1(x)$ and $\zeta_2(u)$ are selected to be linear for the sake of simplicity. Again, the procedures for the algorithm selection and the multiprocessor implementation do not depend crucially on the nature of the elements of the vector b.

5.4 A benchmark for analysis of the multiprocessor implementation

As an example, let us now apply the procedure to generate a synthetic benchmark, variations of which are used for the analysis of the performance of the multiprocessor implementation procedure.

Selection of the Dynamic Structure

The mechanical structure shown in Figure 5.2 is a very simple one, resembling a space shuttle with one flexible arm that supports a solar array. This structure:

1. Consists of nb=8 interconnected bodies.

2. It has a tree type of topology.

3. The bodies can be ordered as {8,7,6,3,5,2,4,1}, according to their frequency content in relation to their mass and elasticity.

4. All bodies are assumed to be unconstrained. Therefore they all have 6 d.o.f..

Dimension of the ODE System

N=48 (=6×8) state variables are needed to describe the dynamics of the structure shown in Figure 5.2.

and the second box corresponds to the block:

$$\begin{bmatrix} 0.002 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.002 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.002 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.002 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.002 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.002 \end{bmatrix}$$

The couplings have been selected so that the sums of the elements of the rows of the matrix F do not exceed ε.

Given the matrix F and the set of eigenvalues, the matrix of the effect ρ_{ij} of an eigenvalue λ_j on a state variable x_i is the matrix of the N eigenvectors of F [Stew73]. Remember that this matrix is needed for the evaluation of the accuracy of the algorithms.

Selection of the Inputs of the System

Suppose that one is interested in the response of the mechanical structure when a sudden excitation applies to body #3. In particular, consider this excitation to be a force in the (x-y) plane of the form of a step function, as shown in Figure 5.2. This can be the case when a force is used to turn the arm of a space shuttle. Then input vector consists of only 2 nonzero elements that correspond to the first 2 d.o.f. of body #3. To assure physical stability a relatively small input, such as 0.05, is considered.

Selection of the Matrix G

Table 5.3 shows the coupling between the bodies and the inputs. G should be a 6×6 block matrix similar to the matrix F. The first box in Table 5.3 for example, corresponds to the block:

Table 5.1: A set of eigenvalues assumed to be contributed by the bodies

body	eigenvalue	frequency
8	-0.025+j314.159	50 Hz
7	-0.025+j125.663	20 Hz
6	-0.025+j50.265	8 Hz
3	-0.025+j31.415	5 Hz
5	-0.025+j18.849	3 Hz
2	-0.025+j12.566	2 Hz
4	-0.025+j3.141	0.5 Hz
1	-0.025+j1.256	0.2 Hz

Selection of the Tolerance ε

ε can be selected to be 0.01 so that the real part of the eigenvalues lies between -0.035 and -0.015. This guarantees that all the components will be damped in 132-307 secs. In addition, the difference between the specified and the actual eigenvalues will be sufficiently small.

Development of the Matrix F

Table 5.2 shows the coupling between the bodies together with the eigenvalues selected. Each of the boxes of this table correspond to a 6×6 block for the matrix F. The first box for example, corresponds to the block:

$$\begin{bmatrix} -0.025 & 1.256 & 0 & 0 & 0 & 0 \\ -1.256 & -0.025 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.025 & 1.256 & 0 & 0 \\ 0 & 0 & -1.256 & -0.025 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.025 & 1.256 \\ 0 & 0 & 0 & 0 & -1.256 & -0.025 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.5 & 1 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 1 & 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 & 1 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 & 0.5 & 1 & 0.5 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 1 \end{bmatrix}$$

and the second box corresponds to the block:

$$\begin{bmatrix} 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \end{bmatrix}$$

Development of the nonlinear matrix $A(x)$

As the mechanical structure has the topology of a tree and the connectivity of the bodies is relatively small, A is sparse. For the sake of simplicity, as explained in Chapter 7, consider a 95% sparsity distributed linearly over the 48 rows as shown in Figure 7.1a. For each nonzero element of the matrix A:

1. The value of the constant η is generated randomly, equally probable to be either positive or negative, and not greater than 5 in absolute value.

2. The matrices $\alpha,\beta,\gamma,\delta$ and ξ are randomly developed with 99% sparsity, as shown in Figure 7.1a, and with values of their nonzero elements equally probable to be either positive or negative, and not greater than 5 in absolute value.

Such an ODE system has nonlinear terms of size similar to the linear ones.

Table 5.2: Eigenvalues and coupling between bodies of the benchmark for analysis of the multiprocessor implementation procedure

	body #1	body #2	body #3	body #4	body #5	body #6	body #7	body #8
body #1	-0.025+j1.256	0.002	0.001	0.0005	0.0001	0	0	0
body #2	0.002	-0.025+j12.566	0.002	0.001	0.0005	0.0001	0	0
body #3	0.001	0.002	-0.025+j31.415	0.002	0.001	0.0005	0.0001	0.0001
body #4	0.0005	0.001	0.002	-0.025+j3.141	0.002	0.001	0.0005	0.0005
body #5	0.0001	0.0005	0.001	0.002	-0.025+j18.849	0.002	0.001	0.001
body #6	0	0.0001	0.0005	0.001	0.002	-0.025+j50.265	0.002	0.002
body #7	0	0	0.0001	0.0005	0.001	0.002	-0.025+j125.663	0.001
body #8	0	0	0.0001	0.0005	0.001	0.002	0.001	-0.025+j314.159

Computation of the Vector u_0

As the two nonzero elements u_{13} and u_{14} of the vector u are step functions, the vector u_0 is identical to u.

Development of the benchmark

Finally the benchmark can be generated following the computations shown in (5.10) and (5.11).

A FORTRAN subroutine that incorporates all the above considerations is presented in Appendix C. This subroutine has been applied on the specific structure discussed above and a complete ODE system, shown in Appendix B, has been generated. Appendix B contains also all the variations of this benchmark that are used for the performance analysis of the multiprocessor implementation procedure.

5.5 A benchmark for analysis of the algorithm selection

The benchmark developed in the previous section is sufficient for the analysis of the performance of the multiprocessor implementation procedure but it is too large for the analysis of the algorithm selection procedure. It has been decided that the analysis of the algorithm selection procedure is based on a smaller structure, a subset of the one shown in Figure 5.2, consisted of:

1. The 6 bodies {1,2,3,6,7,8} numbered as {1,2,3,4,5,6}.

2. Each constrained to move on the (x-y) plane, having therefore only 2 d.o.f..

This structure can be described by N=12 (=6×2) state variables. Tables 5.4 shows the eigenvalues and the coupling between the bodies. Table 5.5 shows the coupling

Table 5.3: Coupling between bodies and inputs of the benchmark for analysis of the multiprocessor implementation procedure

	body #1	body #2	body #3	body #4	body #5	body #6	body #7	body #8
body #1	1(0.5)	0.2	0.1	0	0	0	0	0
body #2	0.2	1(0.5)	0.2	0.1	0	0	0	0
body #3	0.1	0.2	1(0.5)	0.2	0.1	0	0	0
body #4	0	0.1	0.2	1(0.5)	0.2	0.1	0	0
body #5	0	0	0.1	0.2	1(0.5)	0.2	0.1	0.1
body #6	0	0	0	0.1	0.2	1(0.5)	0.2	0.2
body #7	0	0	0	0	0.1	0.2	1(0.5)	0.1
body #8	0	0	0	0	0.1	0.2	0.1	1(0.5)

Table 5.4: Eigenvalues and coupling between bodies of the benchmark for analysis of the algorithm selection procedure

	body #1	body #2	body #3	body #4	body #5	body #6
body #1	-0.025+j1.256	0.002	0.001	0.0005	0.0001	0.0001
body #2	0.002	-0.025+j12.566	0.002	0.001	0.0005	0.0005
body #3	0.001	0.002	-0.025+j31.415	0.002	0.001	0.001
body #4	0.0005	0.001	0.002	-0.025+j50.265	0.002	0.002
body #5	0.0001	0.0005	0.001	0.002	-0.025+j125.663	0.001
body #6	0.0001	0.0005	0.001	0.002	0.001	-0.025+j314.159

Table 5.5: Coupling between bodies and inputs of the benchmark for analysis of the algorithm selection procedure

	body #1	body #2	body #3	body #4	body #5	body #6
body #1	1(0.5)	0.2	0.1	0	0	0
body #2	0.2	1(0.5)	0.2	0.1	0	0
body #3	0.1	0.2	1(0.5)	0.2	0.1	0.1
body #4	0	0.1	0.2	1(0.5)	0.2	0.2
body #5	0	0	0.1	0.2	1(0.5)	0.1
body #6	0	0	0.1	0.2	0.1	1(0.5)

between the bodies and the inputs. Lower sparsities can be used for this benchmark. It has been decided to use:

1. 50% sparsity for the matrix A, distributed linearly over the rows from 42% to 58% similar to Figure 7.1.

2. 90% sparsity for the matrices $\alpha, \beta, \gamma, \delta$ and ξ.

The resulting benchmark is shown in Appendix B together with its variations used for the analysis of the algorithm selection procedure.

5.6 Summary

This chapter is devoted to the selection of an ODE system to serve as a benchmark for the performance evaluation of the algorithm selection and the multiprocessor implementation procedures. These performance analyzes require that the benchmark is generated artificially rather than representing the dynamics of real space structures. It is sufficient to generate a simplified benchmark in the form $A(x)\dot{x}=A(x_0)Fx+A(x_0)Gu$, where the matrices F and G represent the coupling of the state variables x with themselves and with the inputs u respectively, and x_0 is the equilibrium state vector. The frequency and damping of the benchmark is expressed by the 2×2 diagonal blocks of F. The elements of the matrix A include the necessary nonlinearities and are suggested to be generated randomly. A particular benchmark is generated as an example to be used for the analysis of the performance of the multiprocessor implementation procedure. This benchmark consists of 48 ODEs, 90% sparse with relatively simple nonzero terms, describing the dynamics of the structure shown in Figure 5.2. Its eigenvalues are shown in Table 5.1. A subset of this bench-

6. PERFORMANCE OF THE ALGORITHM SELECTION PROCEDURE

This chapter is devoted to the analysis of the performance of the procedure for selection of algorithms that solve ODE systems. The procedure is presented in Chapters 2 and 3. The analysis in this chapter shows the performance of the procedure when applied to variations of the benchmark ODE system which is proposed in Chapter 5. The first section defines the optimality of the selection of algorithms and the objectives of the performance analysis of the procedure. The second section proposes an approach to achieve this objectives as well as an implementation of the procedure in the different test cases. The last section compares the results of the procedure with the actual results obtained by the solution of the benchmarks. This comparison leads to the performance analysis of the procedure.

6.1 Criteria and objectives of the performance analysis

The procedure for the selection of algorithms is presented in Chapters 2 and 3. Figures 2.1 to 2.4 in particular show the steps of the procedure. One can notice from these figures, that the analysis of the performance of the procedure requires only the analysis of the module for the selection of the optimal formula for each group of ODEs of the system. The implementation of this module is shown in Figure 2.4. The optimality of the procedure can be measured by the accuracy of the prediction of the optimal order and step size of the formulas that the module suggests. It should be recognized that the evaluation of the performance of the procedure constitutes essentially a test of the accuracy of the expressions (2.5), (2.10) and (3.5) that are proposed to evaluate the accuracy and the stability of an algorithm.

mark, consisted of 12 ODEs with 50% sparsity, is generated to be used for the analysis of the algorithm selection procedure. Variations of both benchmarks are to be used to test the two procedures.

performance of the procedure can be based on an analysis of its sensitivity to variations of its inputs. It is sufficient to study the range of possible variation of the values of the inputs and focus on those values that may have a significant impact on the performance of the procedure. This subset of input values can define the minimum set of tests required.

According to Figure 2.4, the variable inputs of interest to each formula selection module include the:

1. ODE system
2. Algorithm

Among the characteristics of the ODE system, the selection of algorithms may depend significantly only on the:

1. Size of the longest eigenvalue, or equivalently of the size of the highest frequency.
2. Size of the nonlinear terms relatively to the size of the linear terms.

In Chapter 5 it is explained that the procedure should be evaluated for:

1. Highest frequencies of 50 and 90 Hz.
2. Nonlinear terms of size equal to and one tenth than the size of the linear terms.

An ODE system is proposed in Chapter 5 to serve as a benchmark. This benchmark:

1. Consists of 12 ODEs.

The objectives of the performance analysis of the algorithm selection procedure are:

1. The prediction of its performance under several conditions of interest. Then characteristics of the procedure can be tuned appropriately to maximize the accuracy of the selection of algorithms.

2. The verification of the procedure.

3. An indication of how the procedure could be evaluated for different ODE systems and algorithms.

The performance analysis presented in this chapter is mainly based on the first objective. The achievement of the other objectives is a byproduct of this analysis.

6.2 An approach for the performance analysis

The evaluation of the performance of the algorithm selection procedure can be based on the verification of the optimality of the selection of the orders and step sizes of the candidate algorithms. This can be performed by measuring the actual optimal orders and step sizes and comparing them with those predicted by the procedure. The actual optimal orders and step sizes for each algorithm can be determined by applying again the procedure shown in Figure 2.4 with the accuracy and the stability measured by actually solving the benchmark, rather than from (2.5) and (3.5).

The strategy to analyze the performance of the algorithm selection procedure should combine completeness with simplicity. The smallest but sufficient set of tests, that are as simple as possible, has to be selected. The analysis of the

This grouping is justified by the fact that the highest important frequency F for each group, and equivalently according to (3.1) the maximum step sizes that can be used for each group, differ by one order of magnitude. Table 6.1 demonstrates this fact.

Table 6.1: The groups of ODEs of the benchmark

Group	N	f_{max}	F	h_{max}
1	4	50(90)	50(90)	0.001
2	6	50(90)	8	0.01
3	2	50(90)	0.2	0.1

Chapter 3 indicates that the following two algorithms are top candidates to solve the ODE systems that appear in the simulation of the dynamics of space structures:

1. Explicit Power Series Expansion

2. Adams Bashforth

The algorithm selection procedure should definitely be evaluated for both algorithms. In order to select the optimal orders and step sizes according to Figure 2.4, for each of the two algorithms recognize that:

EPSE algorithm

1. According to Figure 3.2, the minimum order m that could be used is 3.

2. The accuracy E can be evaluated according to (2.5), with N taken from Table 6.1, the β parameter taken from Figure 3.10, and Ψ calculated from (3.5)

2. Its highest frequency is 50Hz.

3. Its nonlinear terms are of size similar to the size of the linear terms.

The set of all the required benchmarks can be developed by varying the above benchmark so that its:

1. Highest frequency is 90 Hz, by specifying in Table 5.1 that the eigenvalue of the body #8 is -0.025+j565.486.

2. The its nonlinear terms are one order of magnitude smaller than the linear terms, by specifying that of each nonzero element of the matrix A the randomly generated nonzero elements of the matrices β, γ, δ and ξ are of absolute value not greater than 0.5.

Referring first to Figure 2.3, for each algorithm one must group the 12 ODEs of the benchmarks. According to the eigenvalues shown in Table 5.1, the ODEs must be grouped as follows:

1. Group #1 consists of 4 ODEs, the 9th-10th and 11th-12th, which correspond to frequency content 20 Hz and 50(90) Hz respectively, i.e. in the range (10-100] Hz.

2. Group #2 consists of 6 ODEs, the 3rd-4th, 5th-6th and 7th-8th, which correspond to frequency content 2 Hz, 5 Hz and 8 Hz respectively, i.e. in the range (1-10] Hz.

3. Group #3 consists of 2 ODEs, the 1st-2nd, which correspond to frequency content 0.5 Hz, i.e. in the range [0.1-1] Hz.

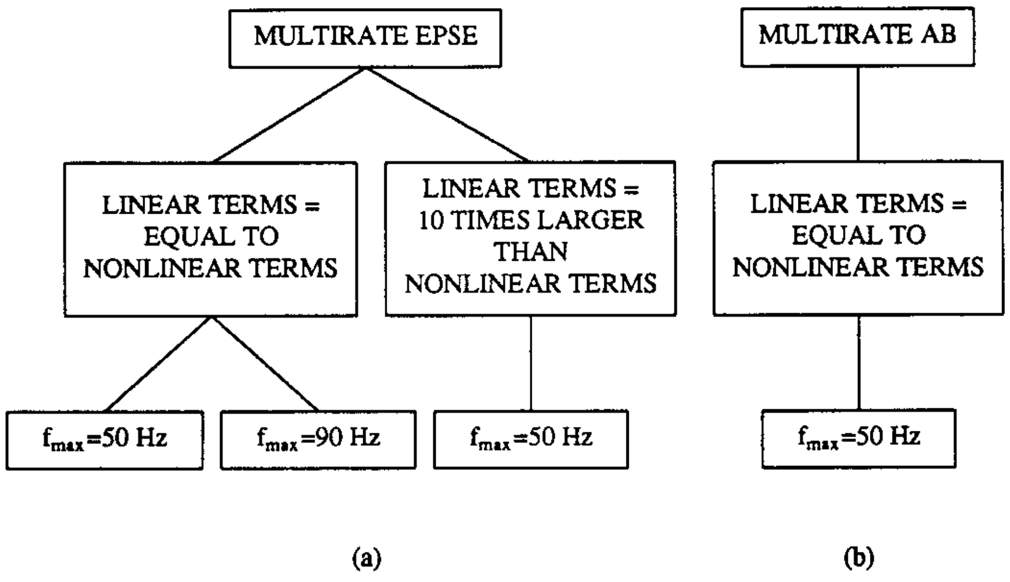

MULTIRATE EPSE
LINEAR TERMS = EQUAL TO NONLINEAR TERMS
LINEAR TERMS = 10 TIMES LARGER THAN NONLINEAR TERMS
f_{max}=50 Hz
f_{max}=90 Hz
f_{max}=50 Hz
MULTIRATE AB
LINEAR TERMS = EQUAL TO NONLINEAR TERMS
f_{max}=50 Hz
(a)
(b)

Figure 6.1: The test cases for the performance analysis of the algorithm selection procedure

The implementation of the two algorithms is shown in Appendix C. The three benchmark ODE systems used by the 4 tests are shown in Appendix B.

6.3 Performance evaluation of the procedure

Consider the application of the algorithm selection procedure, to predict the orders and step sizes of the formulas that the two candidate algorithms should employ in the 4 test cases. Table 6.2 shows the results.

Table 6.3 shows the actual optimal orders and step sizes shown to be required by the experimental solutions of the benchmark. It should be mentioned that 300 with n=300/h points and eigenvalues taken from Table 5.1.

3. The stability Q can be calculated according to (2.10), with the stability region $s(\theta)$ taken from Figure 3.2, and the eigenvalues taken from Table 5.1.

4. The bounds on the accuracy E_u and stability Q_u can be those indicated in (3.6).

AB algorithm

1. According to Figure 3.3, the minimum order m that could be used is 2.

2. The accuracy E can be evaluated according to (2.5), with N taken from Table 6.1, the β parameter taken from Figure 3.10, and Ψ calculated from (3.5) with n=300/h points and eigenvalues taken from Table 5.1.

3. The stability Q can be calculated according to (2.10), with the stability region $s(\theta)$ taken from Figure 3.3, and the eigenvalues taken from Table 5.1.

4. The bounds on the accuracy E_u and stability Q_u can be those indicated in (3.6).

The generation of the set of tests requires the combination of the above different values of its inputs. Each user can select the combinations of its interest. The consideration of all the combinations would result in 36 cases. To avoid performing these many tests, a sufficient subset has to be selected. The 4 tests shown in Figure 6.1 are sufficient to demonstrate the performance of the procedure. The performance of the procedure in the cases that are not considered can be easily implied from the results of these 4 tests.

2. A short period of the solution is obtained.

and does not imply that the procedure is too inaccurate. In addition, the prediction of the step sizes for the EPSE algorithm are less accurate than those for the AB. This is because the EPSE algorithm is implemented in a multirate scheme while the AB algorithm is not. The algorithm selection procedure does not take under account the loss of accuracy and stability caused by the multirate implementation. Such losses are too complicated to formulate. This weekness of the procedure is definitely a point offered for further improvement studies.

A more detailed comparison of the Tables 6.2 and 6.3 indicates that:

1. In the case of 90 Hz maximum frequency the procedure is less accurate than that in the case of 50 Hz. This implies that closer to bounding frequencies, such as 1 Hz, 10 Hz or 100 Hz, the predictions of the procedure, particularly the prediction of the orders, may be less accurate. Nevertheless, the loss of accuracy is not significant.

2. In the case of larger nonlinearities the predictions of the procedure are less accurate. However, the loss of accuracy again is insignificant for an increase of the size of the nonlinearities that does not exceed few orders of magnitude.

In general, the predictions of the procedure although not optimal they are not excessively inaccurate. Realize that, as explained in Chapter 2, in practice the determination of the optimal orders and step sizes requires the experimental solution of the ODE system. The purpose of the algorithm selection procedure is to provide the users with good initial estimation of the optimal orders and step sizes. The results of Table 6.2 are encouraging enough for this purpose.

Table 6.2: Proposed orders and step sizes of the candidate algorithms

Algorithm	Linear terms	f_{max}	Group	order	step size
EPSE	equal to nonlinear terms	50 Hz	1	8	0.001
			2	16	0.01
			3	16	0.1
	10 times larger than nonlinear	50 Hz	1	8	0.001
			2	16	0.01
			3	16	0.1
	10 times larger than nonlinear	90 Hz	1	8	0.001
			2	16	0.01
			3	16	0.1
AB	equal to nonlinear	50 Hz	1	6	0.0001
			2	6	0.0001
			3	6	0.0001

secs long solutions were experienced to be too expensive computationally. Solutions with a relatively large step size of the order of 0.001 require the computation of 300,000 points. As a compromise between reality and simplicity it has been decided to compute solutions as long as 10 secs only corresponding to 10,000 points. A comparison with the Table 6.2 indicates that in general the orders and the step sizes proposed by the algorithm selection procedure are a little conservative. However, this is due to the facts that:

1. The benchmarks are relatively small and simple.

of the module of the prediction of the optimal orders and step sizes to be used for the solution of the different groups of ODEs of the system. This is performed by comparing, in each of the test cases, the orders and step sizes that the procedure suggests with the optimal ones. The optimal orders and step sizes are measured by solving the benchmarks with each of the candidate algorithms repetitively until solutions with acceptable accuracy and stability are obtained. Tables 6.2 and 6.3 show the proposed and the optimal orders and step sizes. Their comparison indicates that the prediction of the procedure is fairly accurate.

Table 6.3: Optimal orders and step sizes of the candidate algorithms

Algorithm	Linear terms	f_{max}	Group	order	step size
EPSE	equal to nonlinear terms	50 Hz	1	6	0.001
			2	12	0.01
			3	12	0.1
	10 times larger than nonlinear	50 Hz	1	7	0.001
			2	12	0.01
			3	12	0.01
	10 times larger than nonlinear	90 Hz	1	7	0.001
			2	12	0.01
			3	12	0.01
AB	equal to nonlinear	50 Hz	1	4	0.001
			2	4	0.001
			3	4	0.001

6.4 Summary

This chapter is devoted to the performance analysis of the procedure which selects algorithms to solve ODE systems that appear in dynamic simulation of space structures. The procedure is presented in Chapters 2 and 3. Its performance is analyzed through a series of tests defined in Figure 6.1. These tests evaluate the procedure when applied on variations of the benchmark ODE system proposed in Chapter 5, solved by the two algorithms that are shown in Chapter 3 to be the top candidates. The performance evaluation of the procedure is based on the verification

3. The verification of the procedure.

4. An indication of how the procedure could be evaluated on different multiprocessor computer systems and for different formulations of the ODE systems.

The performance analysis presented in this chapter is mainly based on the first objective. The achievement of the other objectives is a byproduct of this analysis.

The objective of the multiprocessor implementation procedure is:

1. Maximization of the speed of the solution of the ODE system. According to Section 6.1, the maximization of the solution speed, and equivalently the performance of the procedure, can be measured by:

    a. The degree of balance of the distribution of the computational load among the processors.

    b. The ratio of the busy to idle time periods of the processors.

2. Maximization of the accuracy in the prediction of the optimal number of processors. This is guaranteed by the fact that measurements of the actual performance of the procedure are taken in the module of ESTIMATION OF THE TIME TO SOLVE THE ODE SYSTEM.

7.2 An approach for the performance analysis of the procedure

The strategy to analyze the performance of the multiprocessor implementation procedure should combine completeness with simplicity. The smallest but sufficient set of tests, that are as simple as possible, has to be selected. The analysis of the performance of the procedure can be based on an analysis of its sensitivity to

7. PERFORMANCE OF THE MULTIPROCESSOR IMPLEMENTATION PROCEDURE

This chapter is devoted to the analysis of the performance of the procedure for the implementation of algorithms that solve systems of ODEs on multiprocessor computers. The procedure is presented in the previous chapter. The first section defines the objectives of the performance analysis and the criteria for optimality of the procedure. The second section presents the approach that is followed for the performance analysis. The following section defines the sufficient set of tests required for the analysis of the performance. The results of these tests are presented and discussed in the final section.

7.1 Objectives and criteria of the performance analysis

The objectives of the performance analysis of the multiprocessor implementation procedure are:

1. The prediction of its performance under several conditions of interest. Then characteristics of the procedure or the multiprocessor computer can be tuned appropriately to maximize the performance for the solution of a particular ODE system.

2. An estimation and comparison of the speed of algorithms. In particular, it is interesting to compare the EPSE and AB algorithms, that are shown in Chapter 3 to be most advantageous to solve ODE systems that appear in the dynamic simulation of space structures. Such an estimation and comparison is required by the algorithm selection procedure, shown in Figure 2.2.

3. Multiprocessor architecture

The performance analysis of the procedure can be performed through experimentation on appropriate benchmark ODE systems. A set of variations of the benchmark proposed in Chapter 5 are used in this chapter. The set of necessary tests defines the variations that are required.

It is too difficult and time consuming for the scope of this dissertation to analyze the performance of the procedure for a large variety of multiprocessor architectures. It is also difficult and time consuming to analyze the performance of the procedure on real multiprocessor computers. Such performance analysis requires the complete implementation of the procedure, as well as the expertise in programming the computers. It has been decided instead, to select a multiprocessor computer with a representative architecture and develop its computer emulated representation. What is needed in particular, is an emulator that can provide the timing of the execution of an algorithm solving a benchmark and implemented according to the multiprocessor implementation procedure. In addition to simplicity, the approach of emulated representation allows more flexibility as one can tune the parameters of the particular multiprocessor architecture to tailor the requirements of the different tests. It should be recognized that this emulation would essentially be part of the multiprocessor implementation procedure itself, as a tool of the ESTIMATION OF THE TIME TO SOLVE THE ODE SYSTEM module in particular.

The multiprocessor computer CAPPS [Gluc85], developed by Paragon Pacific, and the computer communication simulator NETWORK II.5 [Garr85], developed by C.A.C.I., have been selected for the performance analysis. The CAPPS computer is a highly promising multiprocessor computer specialized for scientific variations of its inputs. It is sufficient to study the range of possible variation of the values of the inputs and focus on those values that may have a significant impact on the performance of the procedure. This subset of input values can define the minimum set of tests required.

The performance of the procedure depends on the multiprocessor architecture as well as on the length and the coupling of the tasks into which the computation of the solution of the ODE system is partitioned. The reasons for the latter are:

1. The assignment of the tasks is expected to result in earlier completion times of the processors if the differences between the lengths of the tasks are relatively small. Small differences between the length of the tasks may also help the FFD algorithm to determine the optimal number of processors more accurately, so that less improvement iterations are required. The partitioning procedure is designed to guarantee that this will always be true.

2. Earlier completion times are expected when the coupling between the tasks that have been assigned to different processors is low. Again, low coupling between the tasks may increase the accuracy of the FFD algorithm as well. The partitioning and the sequencing of the tasks are designed to minimize the impact of this coupling.

The length and the coupling of the tasks depend on the characteristics of the ODE system and the algorithm. Therefore, the inputs to the procedure that may have a significant impact on the performance of the procedure are:

1. ODE system

2. Algorithm

systems, algorithms and multiprocessor architectures. Let us study each of the inputs separately.

ODE system

Chapter 5 presented the development of a benchmark ODE system that consists of 48 ODEs. According to the frequency content of the state variables, specified by Table 5.1, the state vector can be divided into three groups, as shown in Table 7.1. Then the algorithms should be applied in a multirate scheme, meaning that formulas with different step size and different order, are used for the solution of these different groups of ODEs.

Table 7.1: Groups of ODEs of the benchmark for analysis of the multiprocessor implementation procedure

Group	N	f_{max}	F	h_{max}
1	12	50	50	0.001
2	24	50	8	0.01
3	12	50	0.2	0.1

The length and coupling of the tasks that correspond to a particular benchmark ODE system depend on the sparsity of its:

1. Matrix A(x).
2. Matrices $\alpha, \beta, \gamma, \delta$ and ξ for each nonzero element of A(x).
3. Matrices F and G.

computations, based on a message passing architecture. It is described in Appendix B.

NETWORK II.5 is a very high language interactive simulator written in SIMSCRIPT II.5. Based on a very high level language it provides:

1. A very user friendly environment.

2. Less flexibility. However, it is sufficient for our studies because it can provide all the necessary timing information about the execution of the computation involved in the solution of an ODE system.

Appendix B includes a NETWORK II.5 representation of CAPPS in order to demonstrate the use of the NETWORK II.5 simulator and to provide more detailed description of CAPPS.

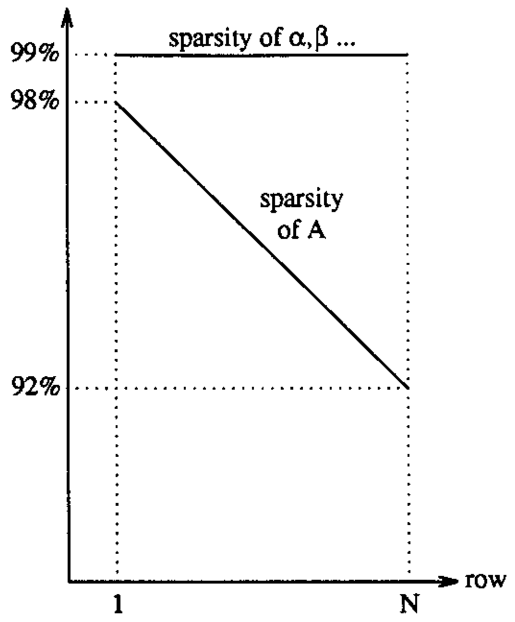
7.3 The selection of the set of tests

The minimum sufficient set of tests has to be selected. The selection of the tests can be based on the determination of the range of variation of the inputs of the procedure that has a significant impact on its performance. In particular, one has to determine the range of variation of:

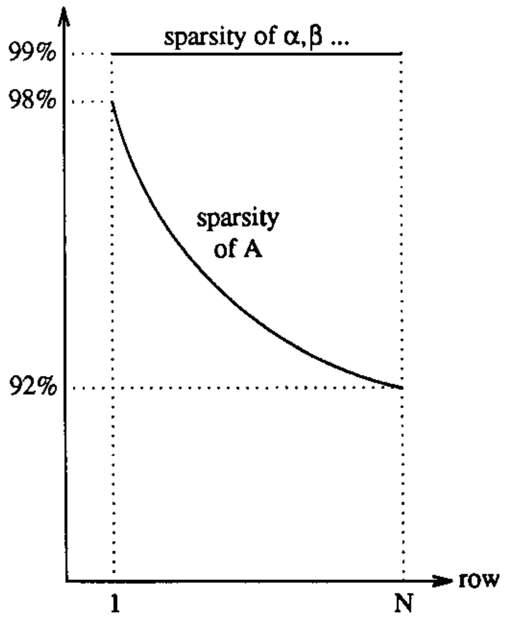
1. The characteristics of the ODE system and the algorithm that produce sets of tasks with a sufficiently wide range of lengths and couplings.

2. The characteristics of the multiprocessor architecture that provide a sufficiently wide range of speeds of execution of the tasks.

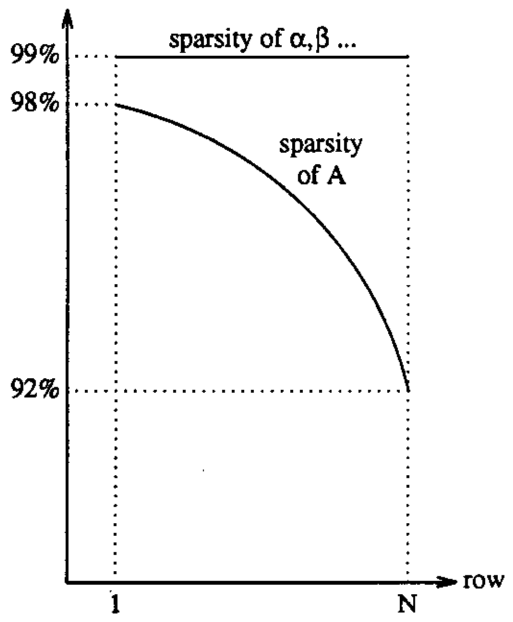
Then, one has to generate the test cases by defining combinations of different ODE



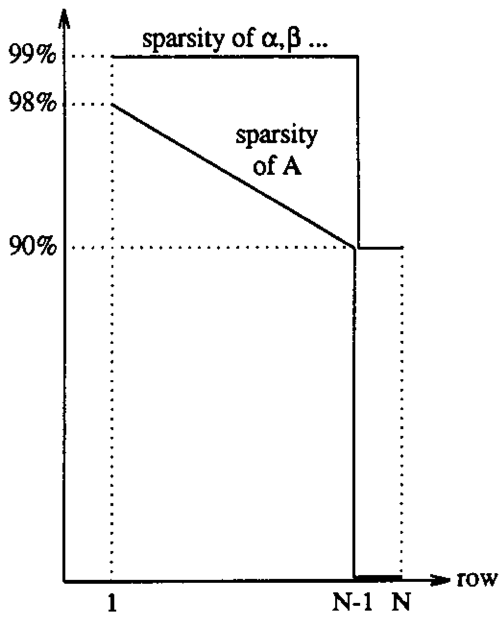
(a)



(b)



(c)



(d)

Figure 7.1: Sparsities of the benchmarks

For the sake of simplicity, one could try to generate benchmarks, that correspond to tasks with lengths and coupling of a variety that is sufficient for the performance analysis, by varying as few of these sparsities as possible. In particular, the impact of the sparsity of the matrices α, β, γ, δ and ξ, as well as of F and G, can be covered in most cases by the impact of the sparsity of A, which is easier to control while developing a benchmark. Then, a constant sparsity can be considered for the matrices of α, β, γ, δ, ξ, F and G, except in the case of producing tasks with very different lengths. Such exception is necessary in case the control of the sparsity of A cannot provide a set of tasks with the desirable differences in length. In addition, the constant sparsity of these matrices can be considered to be such that it results in tasks fully coupled, so that the procedure is tested under worst case conditions.

Figure 7.1 shows a variety of sparsities that can produce the minimum but sufficient variety of sets of tasks required for the performance analysis. All possible kinds of distribution of the sparsities of the rows of matrix A(x) are shown, assuming, without loss of generality, that the rows are sorted according to their length of processing time. The distributions of (a), (b) and (c) correspond to all possible regular cases where no task is much longer than the rest of the tasks. The distribution (d) corresponds to the case that one of the tasks is so much longer than the rest of the tasks, so that it must be cut to three subtasks. Such high sparsities were selected for the sake of simplicity so that none of the resulting tasks exceeds the bound on speed, which is decided to be the real time speed. The large size of the sparsities is acceptable because the performance of the procedure depends on the relative size of the sparsities rather than on the absolute one. Therefore, the multiprocessor implementation procedure must be tested for four different benchmark ODE systems, each defined by one of the four sparsities shown in Figure 7.1.

algorithms with the same number of processors.

The conclusion is that the performance of the procedure should be analyzed for a number of processors:

1. That the procedure itself would suggest.

2. Smaller than the one the procedure would suggest, say half as many.

3. That is the same for EPSE and AB algorithm, say the number that the procedure suggests for the EPSE algorithm.

The generation of the set of tests requires the combination of different values of the inputs in the ranges discussed above. Users can select the combinations of their interest among the above variations of the inputs. The consideration of all the combinations would result in 65 test cases. To avoid performing these many tests, a representative subset has to be selected. The 11 test cases, shown in Figure 7.2, are considered to be sufficient to demonstrate the performance of the procedure. The performance of the cases that are not considered, can be easily implied from the performance of the cases of Figure 7.2.

7.4 Performance analysis of the procedure

Table 7.2 summarizes the results of the 11 tests specified in Figure 7.2. These results are generally very encouraging as they indicate that the performance of the procedure is excellent in all cases. They indicate that the performance of the procedure does not depend on the sparsities of the ODE system. They also indicate that the performance of the procedure is worse for larger number of processors. This

The algorithm

For the performance analysis of the procedure one should select algorithms whose characteristics could result in sets of tasks with lengths and couplings of a sufficiently wide variety. Chapters 3 shows that the EPSE and AB algorithms have significant advantages comparing to other widely used algorithms for the dynamic simulation of space structures. The optimal step sizes and orders of the formulas of the EPSE and AB algorithms, shown in Table 6.3, are used for the performance analysis.

Computer architecture

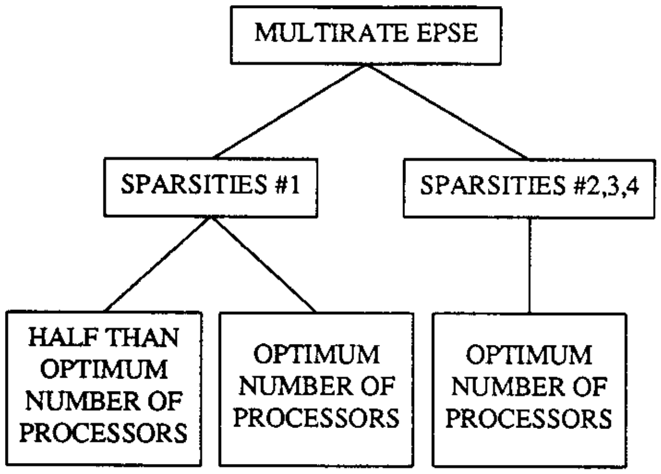
All tests are based on the architecture of the CAPPS computer. For the sake of simplicity, the only architectural characteristic that is varied for the different tests is the number of processors. Other characteristics that may affect the speed of the solution of the ODE system, such as the time required to transfer data or to perform certain arithmetic operations, are considered to be the same for all the tests.

The number of processors is not an input to the procedure. The procedure itself suggests how many processors should be used. However, it can be considered as an internal input. Users that are forced to use a limited number of processors, would be interested in the performance of the procedure for a number of processors that is significantly smaller than the procedure would suggest. It is therefore interesting to evaluate the performance of the procedure for the number of processors that the procedure itself suggests and for a relatively smaller number of processors.

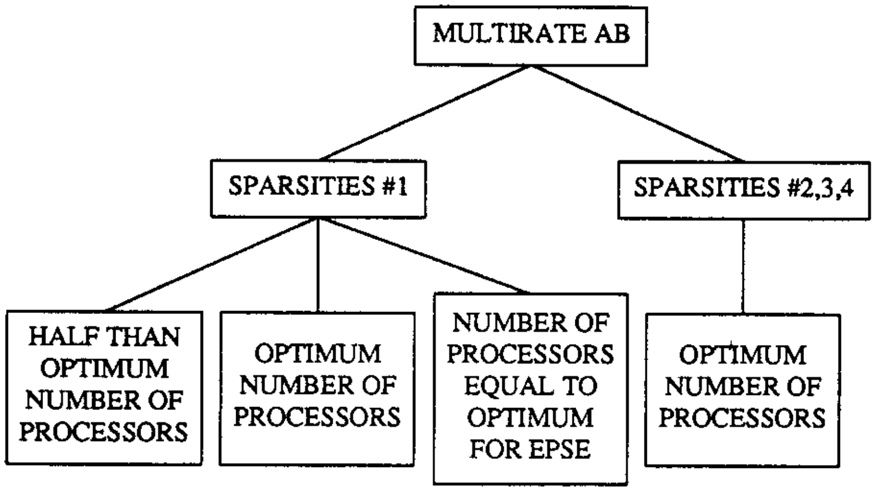
In addition, the objective of comparison of the most competitive algorithms, EPSE and AB in this case, requires the analysis of the procedure employing these should be expected because the distribution of the tasks is more balanced in general when the number of tasks is relatively much larger than the number of processors. For the same reason the performance of the procedure is worse in the case of the EPSE algorithm because the number of processors required is relatively high.

Table 7.2: Performance of the multiprocessor implementation procedure

Algorithm	Sparsity	number of processors	Completion time	Balance	Idle periods	optimal number of processors
EPSE	#1	½ than optimal	1.4 × real time	81%	13%	26
		optimal	1 × real time	53%	23%	
	#2	optimal	1 × real time	49%	29%	22
	#3	optimal	1 × real time	54%	21%	24
	#4	optimal	1 × real time	52%	24%	26
AB	#1	½ than optimal	1.2 × real time	99.5%	0.1%	2
		optimal	0.6 × real time	99%	0.5%	
		optimal for EPSE	0.1 × real time	79%	10%	20
	#2	optimal	0.7 × real time	99%	0.5%	2
	#3	optimal	0.6 × real time	99%	0.5%	2
	#4	optimal	1 × real time	99%	0.5%	2


MULTIRATE EPSE
SPARSITIES #1
SPARSITIES #2,3,4
HALF THAN OPTIMUM NUMBER OF PROCESSORS
OPTIMUM NUMBER OF PROCESSORS
OPTIMUM NUMBER OF PROCESSORS

(a)


MULTIRATE AB
SPARSITIES #1
SPARSITIES #2,3,4
HALF THAN OPTIMUM NUMBER OF PROCESSORS
OPTIMUM NUMBER OF PROCESSORS
NUMBER OF PROCESSORS EQUAL TO OPTIMUM FOR EPSE
OPTIMUM NUMBER OF PROCESSORS

(b)

Figure 7.2: The test cases. Each path of the trees defines a test case

8. CONCLUSION

This final chapter is devoted to a summary and a conclusion of the dissertation. The first section summarizes the accomplishments of the dissertation. The second section indicates the important contributions of the dissertation through a discussion of its strong and week points. The final section suggests possible improvements and complementary studies for the future.

8.1 Summary of the dissertation

The simulation of the dynamics of modern space transportation systems, such as space shuttles and space stations, is an important tool for the improvement of the design of their structure and controls. There is still a great need for improvement of both the computational speed and the cost of such simulations. The major requirement of such a simulation is the long transient part of the solution of a:

1. Large
2. Nonlinear
3. Highly oscillating
4. Lightly damped

system of ODEs that describes the dynamics of the space structures. The complexity of the structures allows only the implicit modeling of the dynamics. The solution of these ODE systems imposes such high computational requirements that:

1. An explicit model should be derived to allow simplifications. In particular, it is suggested that semiexplicit models of the form $A(x,t)\dot{x}=b(x,u,t)$ are derived

The results also indicate that for the particular benchmarks the AB algorithm is faster and requires a smaller number of processors than the EPSE. This should not however considered as a general conclusion. For systems of hundreds of ODEs, as it is mostly the case, the inversion of the matrix A dominates the solution and the AB algorithm is expected to be slower than the EPSE. In addition, the iterative treatment of nonlinearities, an important slow down factor for the AB algorithm, is not considered in the experiments.

7.5 Summary

This chapter is devoted to the analysis of the performance of the procedure for the implementation on multiprocessor computers of algorithms that solve ODE systems. The performance analysis of the procedure is based on the analysis of its sensitivity to variations of the characteristics of the ODE system, of the algorithm and of the number of processors. The minimum sufficient set of 11 tests, shown in Figure 7.2, is selected to perform the performance analysis. The results of these tests, summarized in Table 7.2, indicate that the performance of the procedure is excellent.

perform.

The selection of the algorithm is based on the comparison of the performance of a wide range of algorithms, as shown in Figure 2.1. This comparison requires:

1. The selection of the minimum set of candidate algorithms.

2. The determination of the optimal order and step size of each algorithm, as suggested by Figures 2.3 and 2.4, so that it can meet the minimum speed, accuracy and stability requirements.

3. Evaluation of the performance of each algorithm when implemented in multiprocessor computer with the optimal order and step size, as suggested by Figure 2.2.

The multiprocessor implementation of the algorithm, shown in Figure 4.15, includes:

1. Selection of the number of processors, as suggested by Figure 4.13.

2. Partitioning of the entire computation into a number of tasks, as suggested by Figure 4.6.

3. Assignment of these tasks to the processors, as suggested by Figure 4.10.

4. Scheduling of the execution of these tasks, as suggested by Figure 4.11.

5. Asymptotic improvement based on minimization of the idle time of the processors, as suggested by Figure 4.17, and on tuning the number of processors, as suggested by Figure 4.19.

automatically from implicit ones with the help of symbolic manipulation programs.

2. Multiprocessor computer architectures should be used. Architectures based on the abstract models shown in Figures 1.2 and 1.3, for shared memory or message passing computers respectively, are suggested in particular.

In that case one requires a software system that will convert a high level language description of the explicit mathematical model to an execution code for each processor, as shown in Figure 1.4. The objective of this dissertation is to develop a set of design tools that facilitate the development of such a software system. Such a set of design tools has to include procedures for:

1. Selection of the numerical algorithm to solve the ODE system.

2. Multiprocessor implementation of the selected algorithm.

3. Prediction of the performance of the multiprocessor system.

These tools must be developed according to the criteria of:

1. Minimization of the time required for the solution of the ODE system so that the speed and the accuracy of the solution do not fall below certain lower bounds.

2. Minimization of the cost of the solution of the ODE system. This implies:

    i. Minimization of the number of processors to achieve the required speed.

    ii. Simplicity of the tools so that they are easy to implement and fast to algorithms in a multirate implementation, suggested the optimum orders and step sizes shown in Table 6.2.

2. The experimental solution of the benchmarks with the candidate algorithms suggested the actual optimum orders and step sizes shown in Table 6.3.

The comparison of the Tables 6.2 and 6.3 indicates that the procedure suggested for the selection of algorithms performs with sufficient accuracy. Based on approximations to satisfy the simplicity criterion, it is not expected to provide optimal results in general. It is expected however, to provide a good initial prediction.

The performance of the procedure for the multiprocessor implementation of algorithms can be evaluated by:

1. The completion time of the solution of the ODE system.
2. The degree of balance of the distribution of the computational load among the processors.
3. The relation of the idle and busy periods of the processors.
4. The accuracy of the prediction of the optimum number of processors.

In order to demonstrate and evaluate the procedure, the application of the procedure to a set of 11 test cases, specified in Figure 7.2 by:

1. Four synthetic benchmark ODE systems, specified by different sparsities of the matrices as shown in Figure 7.1.
2. The multirate EPSE and AB algorithms.

The prediction of the performance is suggested:

1. To be a part of the procedure for the multiprocessor implementation of the algorithms, as shown by Figure 4.15.

2. That it can be performed with the help of any commercially available computer system simulator, as shown in Figure 4.16.

The results of a qualitative comparison of the most widely used algorithms, based on an analysis of the computational characteristics of the mathematical models of the dynamics of space structures in relation to the speed, accuracy and stability requirements for the solution of these models, are shown in Table 3.1. These results indicate that the candidate algorithms are:

1. Explicit Power Series Expansion

2. Adams Bashforth

In addition, it is suggested that a multirate implementation of these algorithms can improve their speed, with a tolerable loss of accuracy and stability. Figures 3.11, 3.13 and 3.14 suggest the multirate implementation of the EPSE algorithm. Figures 3.11, 3.13 and 3.12 suggest the multirate implementation of the AB algorithm.

The performance of the procedure for the selection of algorithms can be evaluated by the accuracy of the prediction of the optimum orders and step sizes of the algorithms to be compared. In order to demonstrate and evaluate the procedure:

1. The application of the procedure on three synthetic benchmark ODE systems, specified by different size of nonlinearities and different highest frequency ranges as shown in Figure 6.1, and solved by each of the candidate

Few week aspects of the procedure should be emphasized as well. The measurement of the accuracy of algorithms that the procedure suggests is approximate and the measurement of the stability of the algorithms is based on the equivalent linearized ODE systems. In addition, the loss of accuracy and stability due to the multirate implementation of the algorithms is ignored. As a result, the proposed orders and step sizes might not be the optimal and several trials may be required before the final decisions are maid. Is is however expected that in most cases, the proposed orders and step sizes would be a good starting point for these trials.

The main contribution of the procedure for the multiprocessor implementation of algorithms is the development of a methodology that provides sufficiently optimal multiprocessor implementation of algorithms with a low computational cost. Several of the existing general distributed processing techniques could also be applied. However, the use of such techniques would be wasteful since these techniques being general require a higher computational cost. The procedure suggested in this dissertation constitutes a compromise between optimality and cost. It is specialized to the processing of ODE systems by taking advantage of the specific characteristics of the computation involved and produces results of equivalent performance with a significantly lower cost. The cost savings of the procedure are due to its:

1. Low computational complexity that allows a quick generation of the multiprocessor implementation.

2. Simplicity and modularity that allow easy implementation and modification.

The generality of the procedure is another important advantage of the procedure. Although it emphasizes the solution of ODEs that appear in the simulation of the

3. The implementation of the algorithms on a message passing multiprocessor computer, the CAPPS computer described in Appendix B, emulated with the help of the commercially available computer system simulator NETWORK II.5.

produces the results shown in Table 7.2. These results indicate an excellent performance.

8.2 Evaluation of the dissertation

The research study of the dissertation aims to be a significant contribution to the distributed processing of ODE systems. It is worth to emphasize the uniqueness of this study because in the related fields, such as distributed processing or numerical computing, there has been inadequate concentration on multiprocessor solution of ODEs. As a result the theory and tools developed for ODEs has so far been poor. Due to the great demand for increasing the speed of the solution of ODE systems in numerous industries, the significance of the results of the dissertation is very high.

The main contribution of the procedure for the selection of algorithms is the collection, completion and integration of related theories and techniques into a complete methodology. The important advantages of the procedure include:

1. Simplicity and modularity so that it is easy to implement and modify.
2. Structured and quantitative nature so that precise decisions can be made.
3. Generality so that it can be applied to a wide variety of ODE systems with different characteristics and formulations.

again more expensive, processors or through a larger number of processors. As another example, the experimental application of the procedure can be helpful in selecting between shared memory and message passing multiprocessor architectures for a particular class of applications. In addition, the methodologies involved in the procedures can be helpful in developing distributed system software for multiprocessor computers, such as distributed compilers.

8.3 Suggestions for future complementary studies

In addition to possible improvements of the procedures over the above discussed week points, several complementary studies should be performed in the future to improve the value of these procedures. Among such future studies one should include:

1. The development of techniques for the distributed processing of computations involved in the solution of ODE systems other than the integration of the ODEs. Such computations can be the inversion of matrices, the solution of linear algebraic systems, or the generation of functions. The techniques for distributed processing of these computations should be integrated with the two procedures suggested in the dissertation.

2. The improvement of the applicability of the procedures through a more detailed study of some peculiar situations. For example, there is a need for a more detailed study of the treatment of discontinuities or other types of singularities in the multiprocessor implementation of algorithms.

3. The specifications of the modifications that must be performed on the procedures to make them applicable to other applications of great interest.

dynamics of space structures using either of the candidate algorithms implemented on a CAPPS like architecture, it is sufficiently general to apply and perform well to a wide variety of:

1. ODE systems with different characteristics and formulations.

2. Algorithms

3. Multiprocessor architectures.

without major modifications. The main week aspect of the procedure is the negligence of the loose coupling between the computations involved in the processing of the rows of the ODE systems. The effect of this weekness is expected to be minor in most cases and is attempted to be compensated by the iterative improvement of the multiprocessor implementation.

The generality of both the procedures allows their application to other engineering applications, including various aerospace and robotics applications. The simulation of the dynamics of a helicopter or of the arm of a robot involves characteristics and computational requirements similar to space structures with the major difference being the greater air friction that damps the oscillations faster.

The results of the multiprocessor implementation procedure can also be used to improve the design of multiprocessor architectures to be used for such applications. For example, experimental application of the procedure can show the significance of the performance of the interprocessor communication network relatively to the performance of the processors. This can be valuable information to the architecture designer to help him decide if he should improve the performance of a computer through a faster, but also more expensive, network or through faster, but

References

[Bodl78] Bodley, C.S. *et al.*, "A Digital Computer Program For the Dynamic Interaction Simulation of Controls and Structures (DISCOS), Vol.I and II," Tech. Rep. NASA Technical Report, May 1978.

[Chak83] Chakravarti, P.C. and M.S. Kamel, "Stiffly stable second derivative multistep methods with high order and improved stability regions," *BIT* , bind 23, 1983, p. 75.

[Coff73] Coffman, E.G. and P.J. Denning, *Operating Systems Theory*: Prentice Hall , 1973.

[Coff75] Coffman, E.G. Jr., ed., *Computer and job/shop scheduling theory*: John Wiley & Sons, 1975.

[Coff76] Coffman, E.G. and R. Sethi, "A generalized bound on LPT sequencing," in *Proceedings of the International Symposium on Computer Performance Modeling, Measurement and Evaluation,* Franklin M.A., Ed. March 1976, p. 306.

[Coff78] Coffman, E.G.Jr., M.R. Garey, and D.S. Johnson, "An application of bin-packing to multiprocessor scheduling," *SIAM Journal on Computing,* Vol. 7, No. 1, February 1978, p. 1.

[Cole81] Cole, C.A. et al., *SMP A Symbolic Manipulation Program*, California Institute of Technology, July 1981.

[Dong79] Dongarra, J.J., C.B. Moler, J.R. Bunch, and G.W. Stewart, *LINPACK User's Guide*: SIAM, 1979.

[Enri74] Enright, W.H., "Second derivative multistep methods for stiff ordinary differential equation," *SIAM Journal on Numerical Analysis,* Vol. 11, No. 2, April 1974, p. 321.

[Fern73] Fernandez, E.B. and B. Bussell, "Bounds on the number of processors and time for multiprocessor optimal schedules," *IEEE Transactions on Computers,* Vol. C22, No. 8, August 1973, p. 745.

4. The development of techniques to determine the optimal communication protocol between the processors of a given computer for a specific application. The configuration of the interprocessor communication network and the routing of messages and data through this network are two important problems.

5. The implementation of the procedures in an integrated, general, ready to use software system.

[Ravi86] Ravi, T.M. and M.D. Ercegovac, ''Allocation for the SANDAC Multiprocessor System,'' Tech. Rep. UCLA Computer Science Department Technical Report, February 1986.

[Robi79] Robinson, J.T., ''Some Analysis Techniques for Asynchronous Multiprocessor Algorithms,'' *IEEE Transactions on Software Engineering*, Vol. CE-5, No. 1, January 1979, p. 24.

[Stew73] Stewart, G.W., *Introduction to matrix computations*: Academic Press, Computer Science and Aplied Mathematics , 1973.

[Twiz81] Twizell, E.H. and A.Q.M. Khaliq, ''One step multiderivative method for first order ordinary differential equations,'' *BIT* , bind 21, hefte 4, 1981, p. 518.

APPENDIX A

The multiprocessor computer CAPPS

The multiprocessor computer CAPPS is a loosely coupled message passing distributed system. It was recently developed by Paragon Pacific, El Segundo California, in cooperation with TRW and Northrop. Figure A.1 shows the overall architecture of the CAPPS.

The system contains a number of computational units (CUs), each of which consists of an independent processor with its own memory and control. Each CU is a custom-designed, high performance digital processor. Figure A.2 shows the architecture of a typical CU. This architecture allows the concurrent execution in a pipelined scheme of data and instruction transfers, of arithmetic operations and of input/output operations. The CUs communicate with each other through a network that provides direct connections between input/output queues of the CUs. The configuration and the operation of the network can be optimized for a particular application.

The user controls the system through an initialization and control module (ICM). The ICM is a micro or mini computer with its own memory and input/output capability. The operational programs are stored on floppy disks. and are automatically loaded into the ICM's RAM when the system is powered on. The application programs to be executed by each CU, as well as the required data, are loaded by the

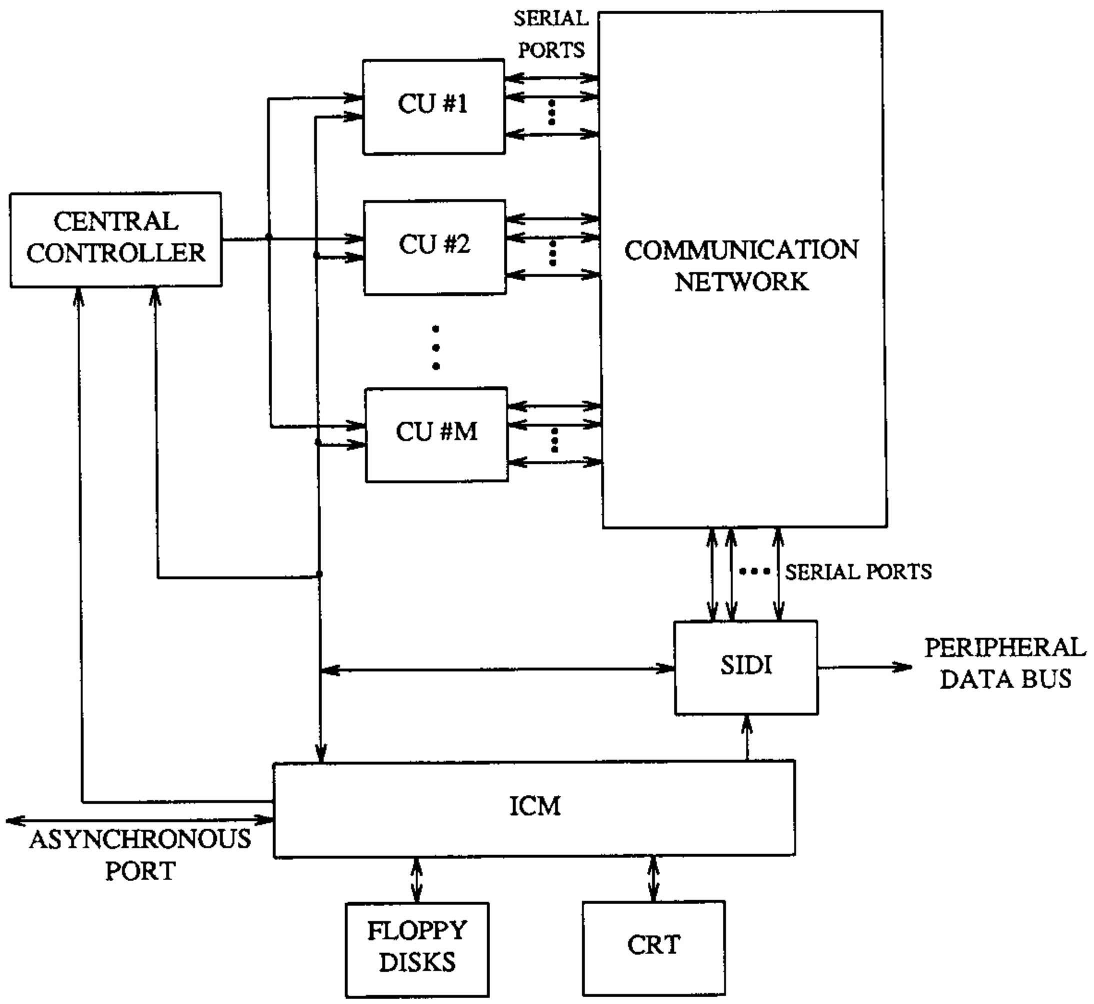


Figure A.1: The overall architecture of CAPPS

ICM into the local memories of the CUs either from floppy disks, or an asynchronous port, or a peripheral bus connected to a system input/output data interphase (SIDI). The system operates under a central controller/sequencer that houses the system clock and logic generators for timing and control of the CUs.

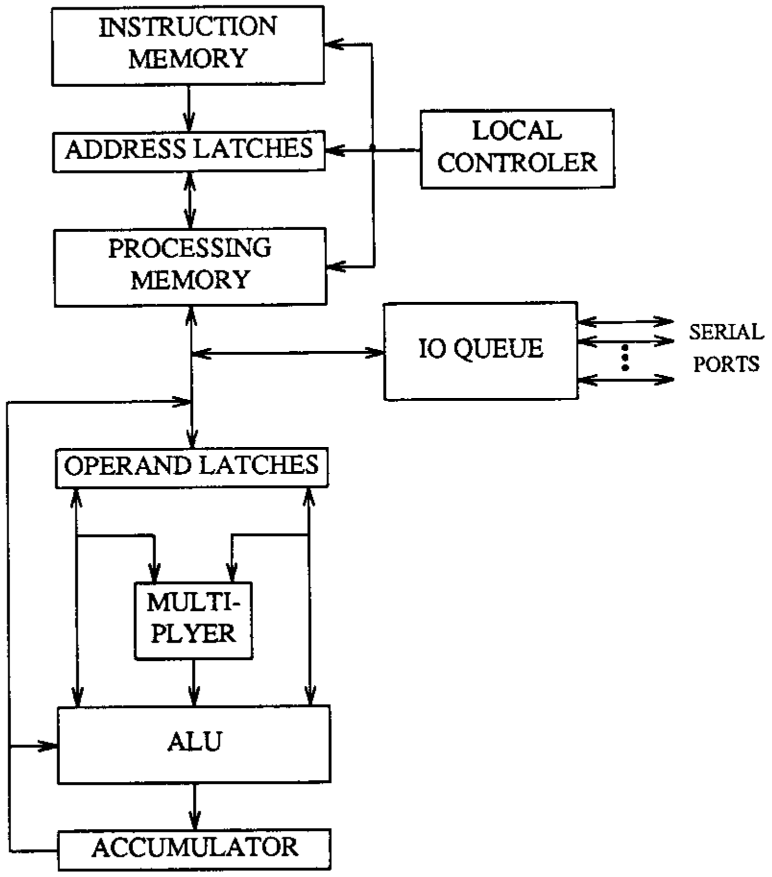


Figure A.2: The architecture of a computational unit of CAPPS

The version of CAPPS available in 1986, the B32, includes twenty 32bit CUs with an instruction cycle of 220 nsecs. More details can be found in [Gluc85].

As explained in Chapter 7, the performance analysis of the procedure for the multiprocessor implementation of algorithms that solve ODE systems requires the emulation of the timing of the solution of the ODE systems by different algorithms on CAPPS. This emulation is performed with the help of the computer system simulator NETWORK II.5. A NETWORK II.5 description of a computer that executes a particular computation consists of the following two parts:

1. Hardware modules that represent the architecture of the hardware devices of the computer.

2. Software modules that represent the computation and the way in which it is going to be executed by the hardware modules.

The program in the following pages is a NETWORK II.5 description of the CAPPS:

1. With 4 identical and synchronized CUs of 250 nsecs instruction cycle each and connected directly to each other with busses of infinite speed.

2. Executing any step of the solution of an ODE system by the single rate EPSE algorithm with order 8.

The hardware modules include:

1. 4 processing elements, named P[1-4], that represent the ALU and memory parts of the CUs.

2. 4 processing elements, named P[1-4]IO, that represent the I/O devices of the CUs.

3. 1 processing element, named CLOCK, that represents a fictitious clock that regulates the I/O queues of the CUs.

The software modules to be executed by these hardware modules include:

1. 9 modules, named INITIALIZATION OF CLOCK, INITIALIZATION OF

P[1-4] and INITIALIZATION OF P[1-4]IO, that trigger the hardware modules to start the execution of their computation.

2. 1 module, named IO CLOCK, that represents the pulse train executed by CLOCK.

3. 32 modules, named D[0-7]ABZ ON P[1-4], that represent the computation of the vectors z for each of the 8 derivatives from (3.14).

4. 4 modules, named DX ON P[1-4], that represent the computation of the derivatives of x from (3.14).

5. 4 modules, named NEW X ON P[1-4], that represent the computation of the vector x_{p+1} from (3.16).

6. 12 modules, named BROADCAST Z FROM P[1-4], BROADCAST DX FROM P[1-4] and BROADCAST NEW X FROM P[1-4], and executed by P[1-4]IO, that represent the interprocessor transfer of the computed elements of the vectors z, the derivatives of x and the vector x_{p+1}.

7. 3 modules, named END BROADCAST Z, END BROADCAST DX and END BROADCAST NEW X, and executed by P1IO, that disable the execution of the above 12 modules.

The execution sequence and dependences of these software modules follows the computational graph of Figure 4.1. Each software module spawns its successors by setting or resetting the proper semaphores. The use of semaphores eliminates the need to emulate the interconnection network and to exchange messages between the processors or the processes.

This is a simplified but fairly sufficient description of CAPPS. It is presented both to describe the operation of CAPPS and to demonstrate the NETWORK II.5 simulator. For more detailed explanation of the following program refer to [Garr85].

```
* CAPPS-4/EPSE-8

***** PROCESSING ELEMENTS - SYS.PE.SET
  HARDWARE TYPE = PROCESSING
   NAME = CLOCK
      BASIC CYCLE TIME =     .220 MICROSEC
      INPUT CONTROLLER = YES
      INSTRUCTION REPERTOIRE =
         INSTRUCTION TYPE = PROCESSING
            NAME ; DELAY
               TIME ;         1 CYCLES
         INSTRUCTION TYPE = SEMAPHORE
            NAME ; IO READY
               SEMAPHORE ; IO READY
               SET/RESET FLAG ; SET
            NAME ; IO NOT READY
               SEMAPHORE ; IO READY
               SET/RESET FLAG ; RESET
            NAME ; CLOCK READY
               SEMAPHORE ; CLOCK READY
               SET/RESET FLAG ; SET
   NAME = P1
      BASIC CYCLE TIME =     .220 MICROSEC
      INPUT CONTROLLER = YES
      INSTRUCTION REPERTOIRE =
         INSTRUCTION TYPE = PROCESSING
            NAME ; D0ABZ ON P1
               TIME ;      4430 CYCLES
            NAME ; D1ABZ ON P1
               TIME ;      7631 CYCLES
            NAME ; D2ABZ ON P1
               TIME ;     10832 CYCLES
            NAME ; D3ABZ ON P1
               TIME ;     14033 CYCLES
            NAME ; D4ABZ ON P1
               TIME ;     17234 CYCLES
            NAME ; D5ABZ ON P1
               TIME ;     20435 CYCLES
            NAME ; D6ABZ ON P1
               TIME ;     23636 CYCLES
            NAME ; D7ABZ ON P1
               TIME ;     26837 CYCLES
            NAME ; DX ON P1
               TIME ;       528 CYCLES
            NAME ; SERIES EVALUATION
               TIME ;        10 CYCLES
         INSTRUCTION TYPE = SEMAPHORE
            NAME ; Z ON P1 READY
               SEMAPHORE ; Z ON P1 READY
               SET/RESET FLAG ; SET
            NAME ; DX ON P1 READY
               SEMAPHORE ; DX ON P1 READY
               SET/RESET FLAG ; SET
            NAME ; DX FROM P1 READY
               SEMAPHORE ; DX ON P1 READY
               SET/RESET FLAG ; SET
            NAME ; DX ON P1 NOT READY
               SEMAPHORE ; DX ON P1 READY
               SET/RESET FLAG ; RESET
         NAME ; NEW X ON P1 READY
            SEMAPHORE ; NEW X ON P1 READY
            SET/RESET FLAG ; SET
         NAME ; NEW X FROM P1 READY
            SEMAPHORE ; NEW X FROM P1 READY
            SET/RESET FLAG ; SET
         NAME ; NEW X ON P1 NOT READY
            SEMAPHORE ; NEW X ON P1 READY
            SET/RESET FLAG ; RESET
         NAME ; NEW X FROM P2 RECEIVED
            SEMAPHORE ; NEW X FROM P2 RECEIVED
            SET/RESET FLAG ; SET
         NAME ; NEW X FROM P3 RECEIVED
            SEMAPHORE ; NEW X FROM P3 RECEIVED
            SET/RESET FLAG ; SET
         NAME ; NEW X FROM P4 RECEIVED
            SEMAPHORE ; NEW X FROM P4 RECEIVED
            SET/RESET FLAG ; SET
         NAME ; P1 FINISHED
            SEMAPHORE ; P1 FINISHED
            SET/RESET FLAG ; SET
NAME = P1IO
   BASIC CYCLE TIME =     .220 MICROSEC
   INPUT CONTROLLER = YES
   INSTRUCTION REPERTOIRE =
      INSTRUCTION TYPE = PROCESSING
         NAME ; DELAY
            TIME ;          1 CYCLES
      INSTRUCTION TYPE = SEMAPHORE
         NAME ; Z FROM P1 RECEIVED
            SEMAPHORE ; Z FROM P1 RECEIVED
            SET/RESET FLAG ; SET
         NAME ; DX FROM P1 RECEIVED
            SEMAPHORE ; DX FROM P1 RECEIVED
            SET/RESET FLAG ; SET
         NAME ; NEW X FROM P1 RECEIVED
            SEMAPHORE ; NEW X FROM P1 RECEIVED
            SET/RESET FLAG ; SET
         NAME ; Z FROM P1 NOT READY
            SEMAPHORE ; Z FROM P1 READY
            SET/RESET FLAG ; RESET
         NAME ; DX FROM P1 NOT READY
            SEMAPHORE ; DX FROM P1 READY
            SET/RESET FLAG ; RESET
         NAME ; NEW X FROM P1 NOT READY
            SEMAPHORE ; NEW X FROM P1 READY
            SET/RESET FLAG ; RESET
         NAME ; Z FROM P1 NOT RECEIVED
            SEMAPHORE ; Z FROM P1 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; DX FROM P1 NOT RECEIVED
            SEMAPHORE ; DX FROM P1 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; NEW X FROM P1 NOT RECEIVED
            SEMAPHORE ; NEW X FROM P1 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; Z FROM P2 NOT RECEIVED
            SEMAPHORE ; Z FROM P2 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; DX FROM P2 NOT RECEIVED
            SEMAPHORE ; DX FROM P2 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; NEW X FROM P2 NOT RECEIVED
            SEMAPHORE ; NEW X FROM P2 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; Z FROM P3 NOT RECEIVED
            SEMAPHORE ; Z FROM P3 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; DX FROM P3 NOT RECEIVED
            SEMAPHORE ; DX FROM P3 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; NEW X FROM P3 NOT RECEIVED
            SEMAPHORE ; NEW X FROM P3 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; Z FROM P4 NOT RECEIVED
            SEMAPHORE ; Z FROM P4 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; DX FROM P4 NOT RECEIVED
            SEMAPHORE ; DX FROM P4 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; NEW X FROM P4 NOT RECEIVED
            SEMAPHORE ; NEW X FROM P4 RECEIVED
            SET/RESET FLAG ; RESET
NAME = P2
   BASIC CYCLE TIME =     .220 MICROSEC
   INPUT CONTROLLER = YES
   INSTRUCTION REPERTOIRE =
      INSTRUCTION TYPE = PROCESSING
         NAME ; D0ABZ ON P2
            TIME ;      4430 CYCLES
         NAME ; D1ABZ ON P2
            TIME ;      7631 CYCLES
         NAME ; D2ABZ ON P2
            TIME ;     10832 CYCLES
         NAME ; D3ABZ ON P2
            TIME ;     14033 CYCLES
         NAME ; D4ABZ ON P2
            TIME ;     17234 CYCLES
         NAME ; D5ABZ ON P2
            TIME ;     20435 CYCLES
         NAME ; D6ABZ ON P2
            TIME ;     23636 CYCLES
         NAME ; D7ABZ ON P2
            TIME ;     26837 CYCLES
         NAME ; DX ON P2
            TIME ;       528 CYCLES
         NAME ; SERIES EVALUATION
            TIME ;        10 CYCLES
      INSTRUCTION TYPE = SEMAPHORE
         NAME ; Z ON P2 READY
            SEMAPHORE ; Z ON P2 READY
            SET/RESET FLAG ; SET
         NAME ; DX ON P2 READY
            SEMAPHORE ; DX ON P2 READY
            SET/RESET FLAG ; SET
         NAME ; DX FROM P2 READY
            SEMAPHORE ; DX ON P2 READY
            SET/RESET FLAG ; SET
         NAME ; DX ON P2 NOT READY
            SEMAPHORE ; DX ON P2 READY
            SET/RESET FLAG ; RESET
         NAME ; NEW X ON P2 READY
            SEMAPHORE ; NEW X ON P2 READY
            SET/RESET FLAG ; SET
         NAME ; NEW X FROM P2 READY
            SEMAPHORE ; NEW X FROM P2 READY
            SET/RESET FLAG ; SET
         NAME ; NEW X ON P2 NOT READY
            SEMAPHORE ; NEW X ON P2 READY
            SET/RESET FLAG ; RESET
         NAME ; NEW X FROM P1 RECEIVED
            SEMAPHORE ; NEW X FROM P1 RECEIVED
            SET/RESET FLAG ; SET
         NAME ; NEW X FROM P3 RECEIVED
            SEMAPHORE ; NEW X FROM P3 RECEIVED
            SET/RESET FLAG ; SET
         NAME ; NEW X FROM P4 RECEIVED
            SEMAPHORE ; NEW X FROM P4 RECEIVED
            SET/RESET FLAG ; SET
         NAME ; P2 FINISHED
            SEMAPHORE ; P1 FINISHED
            SET/RESET FLAG ; SET
NAME = P1IO
   BASIC CYCLE TIME =      .220 MICROSEC
   INPUT CONTROLLER = YES
   INSTRUCTION REPERTOIRE =
      INSTRUCTION TYPE = PROCESSING
         NAME ; DELAY
            TIME ;          1 CYCLES
      INSTRUCTION TYPE = SEMAPHORE
         NAME ; Z FROM P2 RECEIVED
            SEMAPHORE ; Z FROM P2 RECEIVED
            SET/RESET FLAG ; SET
         NAME ; DX FROM P2 RECEIVED
            SEMAPHORE ; DX FROM P2 RECEIVED
            SET/RESET FLAG ; SET
         NAME ; NEW X FROM P2 RECEIVED
            SEMAPHORE ; NEW X FROM P2 RECEIVED
            SET/RESET FLAG ; SET
         NAME ; Z FROM P2 NOT READY
            SEMAPHORE ; Z FROM P2 READY
            SET/RESET FLAG ; RESET
         NAME ; DX FROM P2 NOT READY
            SEMAPHORE ; DX FROM P2 READY
            SET/RESET FLAG ; RESET
         NAME ; NEW X FROM P2 NOT READY
            SEMAPHORE ; NEW X FROM P2 READY
            SET/RESET FLAG ; RESET
         NAME ; Z FROM P2 NOT RECEIVED
            SEMAPHORE ; Z FROM P2 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; DX FROM P2 NOT RECEIVED
            SEMAPHORE ; DX FROM P2 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; NEW X FROM P2 NOT RECEIVED
            SEMAPHORE ; NEW X FROM P2 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; Z FROM P1 NOT RECEIVED
            SEMAPHORE ; Z FROM P1 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; DX FROM P1 NOT RECEIVED
            SEMAPHORE ; DX FROM P1 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; NEW X FROM P1 NOT RECEIVED
            SEMAPHORE ; NEW X FROM P1 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; Z FROM P3 NOT RECEIVED
            SEMAPHORE ; Z FROM P3 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; DX FROM P3 NOT RECEIVED
            SEMAPHORE ; DX FROM P3 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; NEW X FROM P3 NOT RECEIVED
            SEMAPHORE ; NEW X FROM P3 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; Z FROM P4 NOT RECEIVED
            SEMAPHORE ; Z FROM P4 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; DX FROM P4 NOT RECEIVED
            SEMAPHORE ; DX FROM P4 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; NEW X FROM P4 NOT RECEIVED
            SEMAPHORE ; NEW X FROM P4 RECEIVED
            SET/RESET FLAG ; RESET
```

```
NAME = P3
   BASIC CYCLE TIME =     .220 MICROSEC
   INPUT CONTROLLER = YES
   INSTRUCTION REPERTOIRE =
      INSTRUCTION TYPE = PROCESSING
         NAME ; D0ABZ ON P3
            TIME ;       4430 CYCLES
         NAME ; D1ABZ ON P3
            TIME ;       7631 CYCLES
         NAME ; D2ABZ ON P3
            TIME ;      10832 CYCLES
         NAME ; D3ABZ ON P3
            TIME ;      14033 CYCLES
         NAME ; D4ABZ ON P3
            TIME ;      17234 CYCLES
         NAME ; D5ABZ ON P3
            TIME ;      20435 CYCLES
         NAME ; D6ABZ ON P3
            TIME ;      23636 CYCLES
         NAME ; D7ABZ ON P3
            TIME ;      26837 CYCLES
         NAME ; DX ON P3
            TIME ;        528 CYCLES
         NAME ; SERIES EVALUATION
            TIME ;         10 CYCLES
      INSTRUCTION TYPE = SEMAPHORE
         NAME ; Z ON P3 READY
            SEMAPHORE ; Z ON P3 READY
            SET/RESET FLAG ; SET
         NAME ; DX ON P3 READY
            SEMAPHORE ; DX ON P3 READY
            SET/RESET FLAG ; SET
         NAME ; DX FROM P3 READY
            SEMAPHORE ; DX ON P3 READY
            SET/RESET FLAG ; SET
         NAME ; DX ON P3 NOT READY
            SEMAPHORE ; DX ON P3 READY
            SET/RESET FLAG ; RESET
         NAME ; NEW X ON P3 READY
            SEMAPHORE ; NEW X ON P3 READY
            SET/RESET FLAG ; SET
         NAME ; NEW X FROM P3 READY
            SEMAPHORE ; NEW X FROM P3 READY
            SET/RESET FLAG ; SET
         NAME ; NEW X ON P3 NOT READY
            SEMAPHORE ; NEW X ON P3 READY
            SET/RESET FLAG ; RESET
         NAME ; NEW X FROM P2 RECEIVED
            SEMAPHORE ; NEW X FROM P2 RECEIVED
            SET/RESET FLAG ; SET
         NAME ; NEW X FROM P1 RECEIVED
            SEMAPHORE ; NEW X FROM P1 RECEIVED
            SET/RESET FLAG ; SET
         NAME ; NEW X FROM P4 RECEIVED
            SEMAPHORE ; NEW X FROM P4 RECEIVED
            SET/RESET FLAG ; SET
         NAME ; P3 FINISHED
            SEMAPHORE ; P3 FINISHED
            SET/RESET FLAG ; SET
NAME = P3IO
   BASIC CYCLE TIME =     .220 MICROSEC
   INPUT CONTROLLER = YES
   INSTRUCTION REPERTOIRE =
      INSTRUCTION TYPE = PROCESSING
         NAME ; DELAY
            TIME ;          1 CYCLES
      INSTRUCTION TYPE = SEMAPHORE
         NAME ; Z FROM P3 RECEIVED
```

```
            SEMAPHORE ; Z FROM P3 RECEIVED
            SET/RESET FLAG ; SET
         NAME ; DX FROM P3 RECEIVED
            SEMAPHORE ; DX FROM P3 RECEIVED
            SET/RESET FLAG ; SET
         NAME ; NEW X FROM P3 RECEIVED
            SEMAPHORE ; NEW X FROM P3 RECEIVED
            SET/RESET FLAG ; SET
         NAME ; Z FROM P3 NOT READY
            SEMAPHORE ; Z FROM P3 READY
            SET/RESET FLAG ; RESET
         NAME ; DX FROM P3 NOT READY
            SEMAPHORE ; DX FROM P3 READY
            SET/RESET FLAG ; RESET
         NAME ; NEW X FROM P3 NOT READY
            SEMAPHORE ; NEW X FROM P3 READY
            SET/RESET FLAG ; RESET
         NAME ; Z FROM P3 NOT RECEIVED
            SEMAPHORE ; Z FROM P3 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; DX FROM P3 NOT RECEIVED
            SEMAPHORE ; DX FROM P3 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; NEW X FROM P3 NOT RECEIVED
            SEMAPHORE ; NEW X FROM P3 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; Z FROM P2 NOT RECEIVED
            SEMAPHORE ; Z FROM P2 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; DX FROM P2 NOT RECEIVED
            SEMAPHORE ; DX FROM P2 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; NEW X FROM P2 NOT RECEIVED
            SEMAPHORE ; NEW X FROM P2 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; Z FROM P1 NOT RECEIVED
            SEMAPHORE ; Z FROM P1 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; DX FROM P1 NOT RECEIVED
            SEMAPHORE ; DX FROM P1 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; NEW X FROM P1 NOT RECEIVED
            SEMAPHORE ; NEW X FROM P1 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; Z FROM P4 NOT RECEIVED
            SEMAPHORE ; Z FROM P4 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; DX FROM P4 NOT RECEIVED
            SEMAPHORE ; DX FROM P4 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; NEW X FROM P4 NOT RECEIVED
            SEMAPHORE ; NEW X FROM P4 RECEIVED
            SET/RESET FLAG ; RESET
NAME = P4
   BASIC CYCLE TIME =     .220 MICROSEC
   INPUT CONTROLLER = YES
   INSTRUCTION REPERTOIRE =
      INSTRUCTION TYPE = PROCESSING
         NAME ; D0ABZ ON P4
            TIME ;      4430 CYCLES
         NAME ; D1ABZ ON P4
            TIME ;      7631 CYCLES
         NAME ; D2ABZ ON P4
            TIME ;     10832 CYCLES
         NAME ; D3ABZ ON P4
            TIME ;     14033 CYCLES
         NAME ; D4ABZ ON P4
            TIME ;     17234 CYCLES
         NAME ; D5ABZ ON P4
            TIME ;     20435 CYCLES
         NAME ; D6ABZ ON P4
            TIME ;     23636 CYCLES
         NAME ; D7ABZ ON P4
            TIME ;     26837 CYCLES
         NAME ; DX ON P4
            TIME ;       528 CYCLES
         NAME ; SERIES EVALUATION
            TIME ;        10 CYCLES
      INSTRUCTION TYPE = SEMAPHORE
         NAME ; Z ON P4 READY
            SEMAPHORE ; Z ON P4 READY
            SET/RESET FLAG ; SET
         NAME ; DX ON P4 READY
            SEMAPHORE ; DX ON P4 READY
            SET/RESET FLAG ; SET
         NAME ; DX FROM P4 READY
            SEMAPHORE ; DX ON P4 READY
            SET/RESET FLAG ; SET
         NAME ; DX ON P4 NOT READY
            SEMAPHORE ; DX ON P4 READY
            SET/RESET FLAG ; RESET
         NAME ; NEW X ON P4 READY
            SEMAPHORE ; NEW X ON P4 READY
            SET/RESET FLAG ; SET
         NAME ; NEW X FROM P4 READY
            SEMAPHORE ; NEW X FROM P4 READY
            SET/RESET FLAG ; SET
         NAME ; NEW X ON P4 NOT READY
            SEMAPHORE ; NEW X ON P4 READY
            SET/RESET FLAG ; RESET
         NAME ; NEW X FROM P2 RECEIVED
            SEMAPHORE ; NEW X FROM P2 RECEIVED
            SET/RESET FLAG ; SET
         NAME ; NEW X FROM P3 RECEIVED
            SEMAPHORE ; NEW X FROM P3 RECEIVED
            SET/RESET FLAG ; SET
         NAME ; NEW X FROM P1 RECEIVED
            SEMAPHORE ; NEW X FROM P1 RECEIVED
            SET/RESET FLAG ; SET
         NAME ; P4 FINISHED
            SEMAPHORE ; P4 FINISHED
            SET/RESET FLAG ; SET
NAME = P4IO
   BASIC CYCLE TIME =     .220 MICROSEC
   INPUT CONTROLLER = YES
   INSTRUCTION REPERTOIRE =
      INSTRUCTION TYPE = PROCESSING
         NAME ; DELAY
            TIME ;         1 CYCLES
      INSTRUCTION TYPE = SEMAPHORE
         NAME ; Z FROM P4 RECEIVED
            SEMAPHORE ; Z FROM P4 RECEIVED
            SET/RESET FLAG ; SET
         NAME ; DX FROM P4 RECEIVED
            SEMAPHORE ; DX FROM P4 RECEIVED
            SET/RESET FLAG ; SET
         NAME ; NEW X FROM P4 RECEIVED
            SEMAPHORE ; NEW X FROM P4 RECEIVED
            SET/RESET FLAG ; SET
         NAME ; Z FROM P4 NOT READY
            SEMAPHORE ; Z FROM P4 READY
            SET/RESET FLAG ; RESET
         NAME ; DX FROM P4 NOT READY
            SEMAPHORE ; DX FROM P4 READY
            SET/RESET FLAG ; RESET
         NAME ; NEW X FROM P4 NOT READY
```

```
            SEMAPHORE ; NEW X FROM P4 READY
            SET/RESET FLAG ; RESET
         NAME ; Z FROM P4 NOT RECEIVED
            SEMAPHORE ; Z FROM P4 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; DX FROM P4 NOT RECEIVED
            SEMAPHORE ; DX FROM P4 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; NEW X FROM P4 NOT RECEIVED
            SEMAPHORE ; NEW X FROM P4 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; Z FROM P2 NOT RECEIVED
            SEMAPHORE ; Z FROM P2 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; DX FROM P2 NOT RECEIVED
            SEMAPHORE ; DX FROM P2 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; NEW X FROM P2 NOT RECEIVED
            SEMAPHORE ; NEW X FROM P2 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; Z FROM P3 NOT RECEIVED
            SEMAPHORE ; Z FROM P3 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; DX FROM P3 NOT RECEIVED
            SEMAPHORE ; DX FROM P3 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; NEW X FROM P3 NOT RECEIVED
            SEMAPHORE ; NEW X FROM P3 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; Z FROM P1 NOT RECEIVED
            SEMAPHORE ; Z FROM P1 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; DX FROM P1 NOT RECEIVED
            SEMAPHORE ; DX FROM P1 RECEIVED
            SET/RESET FLAG ; RESET
         NAME ; NEW X FROM P1 NOT RECEIVED
            SEMAPHORE ; NEW X FROM P1 RECEIVED
            SET/RESET FLAG ; RESET

***** MODULES - SYS.MODULE.SET
  SOFTWARE TYPE = MODULE
   NAME = INITIALIZATION OF CLOCK
      INTERRUPTABILITY FLAG = NO
      CONCURRENT EXECUTION = NO
      START TIME =      0.
      ALLOWED PROCESSORS =
         CLOCK
      REQUIRED HARDWARE STATUS =
         CLOCK
      INSTRUCTION LIST =
         EXECUTE A TOTAL OF ;      1 CLOCK READY
      ANDED SUCCESSORS =
         CHAIN TO ; IO CLOCK
         WITH ITERATIONS THEN SKIP COUNT OF ;      0
   NAME = INITIALIZATION OF P1
      INTERRUPTABILITY FLAG = NO
      CONCURRENT EXECUTION = NO
      START TIME =      0.
      ALLOWED PROCESSORS =
         P1
      REQUIRED HARDWARE STATUS =
         P1
      INSTRUCTION LIST =
         EXECUTE A TOTAL OF ;      1 NEW X ON P1 READY
         EXECUTE A TOTAL OF ;      1 NEW X FROM P1 RECEIVED
      ANDED SUCCESSORS =
         CHAIN TO ; DOABZ ON P1
         WITH ITERATIONS THEN SKIP COUNT OF ;      0
```

```
NAME = INITIALIZATION OF P2
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =       0.
   ALLOWED PROCESSORS =
      P2
   REQUIRED HARDWARE STATUS =
      P2
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;       1 NEW X ON P2 READY
      EXECUTE A TOTAL OF ;       1 NEW X FROM P2 RECEIVED
   ANDED SUCCESSORS =
      CHAIN TO ; DOABZ ON P2
      WITH ITERATIONS THEN SKIP COUNT OF ;        0
NAME = INITIALIZATION OF P3
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =       0.
   ALLOWED PROCESSORS =
      P3
   REQUIRED HARDWARE STATUS =
      P3
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;       1 NEW X ON P3 READY
      EXECUTE A TOTAL OF ;       1 NEW X FROM P3 RECEIVED
   ANDED SUCCESSORS =
      CHAIN TO ; DOABZ ON P3
      WITH ITERATIONS THEN SKIP COUNT OF ;        0
NAME = INITIALIZATION OF P4
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =       0.
   ALLOWED PROCESSORS =
      P4
   REQUIRED HARDWARE STATUS =
      P4
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;       1 NEW X ON P4 READY
      EXECUTE A TOTAL OF ;       1 NEW X FROM P4 RECEIVED
   ANDED SUCCESSORS =
      CHAIN TO ; DOABZ ON P4
      WITH ITERATIONS THEN SKIP COUNT OF ;        0
NAME = INITIALIZATION OF P1IO
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =       0.
   ALLOWED PROCESSORS =
      P1IO
   REQUIRED HARDWARE STATUS =
      P1IO
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;       1 Z ON P1 NOT READY
      EXECUTE A TOTAL OF ;       1 NEW X ON P1 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; BROADCAST Z FROM P1
      WITH ITERATIONS THEN SKIP COUNT OF ;        0
      CHAIN TO ; BROADCAST NEW X FROM P1
      WITH ITERATIONS THEN SKIP COUNT OF ;        0
NAME = INITIALIZATION OF P2IO
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =       0.
   ALLOWED PROCESSORS =
      P2IO
   REQUIRED HARDWARE STATUS =
      P2IO
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;       1 Z ON P2 NOT READY
      EXECUTE A TOTAL OF ;       1 NEW X ON P2 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; BROADCAST Z FROM P2
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
      CHAIN TO ; BROADCAST NEW X FROM P2
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
NAME = INITIALIZATION OF P3IO
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =       0.
   ALLOWED PROCESSORS =
      P3IO
   REQUIRED HARDWARE STATUS =
      P3IO
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;       1 Z ON P3 NOT READY
      EXECUTE A TOTAL OF ;       1 NEW X ON P3 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; BROADCAST Z FROM P3
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
      CHAIN TO ; BROADCAST NEW X FROM P3
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
NAME = INITIALIZATION OF P4IO
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =       0.
   ALLOWED PROCESSORS =
      P4IO
   REQUIRED HARDWARE STATUS =
      P4IO
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;       1 Z ON P4 NOT READY
      EXECUTE A TOTAL OF ;       1 NEW X ON P4 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; BROADCAST Z FROM P4
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
      CHAIN TO ; BROADCAST NEW X FROM P4
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
NAME = IO CLOCK
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   ITERATION PERIOD =       4.00 MICROSEC
   START TIME =       0.
   ALLOWED PROCESSORS =
      CLOCK
   REQUIRED HARDWARE STATUS =
      CLOCK
   ORED PREDECESSOR LIST =
      INITIALIZATION OF CLOCK
      IO CLOCK
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;       1 IO READY
      EXECUTE A TOTAL OF ;       1 DELAY
      EXECUTE A TOTAL OF ;       1 IO NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; IO CLOCK
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
NAME = DOABZ ON P1
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =       0.
   ALLOWED PROCESSORS =
      P1
   REQUIRED HARDWARE STATUS =
      P1
   ORED PREDECESSOR LIST =
      INITIALIZATION OF P1
      NEW X ON P1
      REQUIRED SEMAPHORE STATUS =
         WAIT FOR ; NEW X ON P1 READY
         TO BE ; SET
         WAIT FOR ; NEW X FROM P2 RECEIVED
         TO BE ; SET
         WAIT FOR ; NEW X FROM P3 RECEIVED
         TO BE ; SET
         WAIT FOR ; NEW X FROM P4 RECEIVED
         TO BE ; SET
      INSTRUCTION LIST =
         EXECUTE A TOTAL OF ;       1 D0ABZ ON P1
         EXECUTE A TOTAL OF ;       1 Z ON P1 READY
         EXECUTE A TOTAL OF ;       1 NEW X ON P1 NOT READY
      ANDED SUCCESSORS =
         CHAIN TO ; D1ABZ ON P1
         WITH ITERATIONS THEN SKIP COUNT OF ;       0
         CHAIN TO ; DX ON P1
         WITH ITERATIONS THEN SKIP COUNT OF ;       0
   NAME = D0ABZ ON P2
      INTERRUPTABILITY FLAG = NO
      CONCURRENT EXECUTION = NO
      START TIME =      0.
      ALLOWED PROCESSORS =
         P2
      REQUIRED HARDWARE STATUS =
         P2
      ORED PREDECESSOR LIST =
         INITIALIZATION OF P2
         NEW X ON P2
      REQUIRED SEMAPHORE STATUS =
         WAIT FOR ; NEW X ON P2 READY
         TO BE ; SET
         WAIT FOR ; NEW X FROM P1 RECEIVED
         TO BE ; SET
         WAIT FOR ; NEW X FROM P3 RECEIVED
         TO BE ; SET
         WAIT FOR ; NEW X FROM P4 RECEIVED
         TO BE ; SET
      INSTRUCTION LIST =
         EXECUTE A TOTAL OF ;       1 D0ABZ ON P2
         EXECUTE A TOTAL OF ;       1 Z ON P2 READY
         EXECUTE A TOTAL OF ;       1 NEW X ON P2 NOT READY
      ANDED SUCCESSORS =
         CHAIN TO ; D1ABZ ON P2
         WITH ITERATIONS THEN SKIP COUNT OF ;       0
         CHAIN TO ; DX ON P2
         WITH ITERATIONS THEN SKIP COUNT OF ;       0
   NAME = D0ABZ ON P3
      INTERRUPTABILITY FLAG = NO
      CONCURRENT EXECUTION = NO
      START TIME =      0.
      ALLOWED PROCESSORS =
         P3
      REQUIRED HARDWARE STATUS =
         P3
      ORED PREDECESSOR LIST =
         INITIALIZATION OF P3
         NEW X ON P3
      REQUIRED SEMAPHORE STATUS =
         WAIT FOR ; NEW X ON P3 READY
         TO BE ; SET
         WAIT FOR ; NEW X FROM P1 RECEIVED
         TO BE ; SET
         WAIT FOR ; NEW X FROM P2 RECEIVED
         TO BE ; SET
         WAIT FOR ; NEW X FROM P4 RECEIVED
         TO BE ; SET
      INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;      1 D0ABZ ON P3
      EXECUTE A TOTAL OF ;      1 Z ON P3 READY
      EXECUTE A TOTAL OF ;      1 NEW X ON P3 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; D1ABZ ON P3
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
      CHAIN TO ; DX ON P3
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
NAME = D0ABZ ON P4
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =      0.
   ALLOWED PROCESSORS =
      P4
   REQUIRED HARDWARE STATUS =
      P4
   ORED PREDECESSOR LIST =
      INITIALIZATION OF P4
      NEW X ON P4
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; NEW X ON P4 READY
      TO BE ; SET
      WAIT FOR ; NEW X FROM P1 RECEIVED
      TO BE ; SET
      WAIT FOR ; NEW X FROM P2 RECEIVED
      TO BE ; SET
      WAIT FOR ; NEW X FROM P3 RECEIVED
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;      1 D0ABZ ON P4
      EXECUTE A TOTAL OF ;      1 Z ON P4 READY
      EXECUTE A TOTAL OF ;      1 NEW X ON P4 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; D1ABZ ON P4
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
      CHAIN TO ; DX ON P4
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
NAME = BROADCAST Z FROM P1
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =      0.
   ALLOWED PROCESSORS =
      P1IO
   REQUIRED HARDWARE STATUS =
      P1IO
   ORED PREDECESSOR LIST =
      INITIALIZATION OF P1IO
      BROADCAST DX FROM P1
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; Z ON P1 READY
      TO BE ; SET
      WAIT FOR ; IO READY
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;     32 DELAY
      EXECUTE A TOTAL OF ;      1 Z FROM P1 RECEIVED
      EXECUTE A TOTAL OF ;      1 Z ON P1 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; BROADCAST DX FROM P1
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
      CHAIN TO ; END BROADCAST Z
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
NAME = BROADCAST Z FROM P2
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =      0.
   ALLOWED PROCESSORS =
      P2IO
   REQUIRED HARDWARE STATUS =
      P2IO
   ORED PREDECESSOR LIST =
      INITIALIZATION OF P1IO
      BROADCAST DX FROM P1
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; Z ON P2 READY
      TO BE ; SET
      WAIT FOR ; IO READY
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;     32 DELAY
      EXECUTE A TOTAL OF ;      1 Z FROM P2 RECEIVED
      EXECUTE A TOTAL OF ;      1 Z ON P2 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; BROADCAST DX FROM P2
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
      CHAIN TO ; END BROADCAST Z
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
NAME = BROADCAST Z FROM P3
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =       0.
   ALLOWED PROCESSORS =
      P3IO
   REQUIRED HARDWARE STATUS =
      P3IO
   ORED PREDECESSOR LIST =
      INITIALIZATION OF P3IO
      BROADCAST DX FROM P3
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; Z ON P3 READY
      TO BE ; SET
      WAIT FOR ; IO READY
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;     32 DELAY
      EXECUTE A TOTAL OF ;      1 Z FROM P3 RECEIVED
      EXECUTE A TOTAL OF ;      1 Z ON P3 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; BROADCAST DX FROM P3
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
      CHAIN TO ; END BROADCAST Z
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
NAME = BROADCAST Z FROM P4
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =       0.
   ALLOWED PROCESSORS =
      P4IO
   REQUIRED HARDWARE STATUS =
      P4IO
   ORED PREDECESSOR LIST =
      INITIALIZATION OF P4IO
      BROADCAST DX FROM P4
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; Z ON P4 READY
      TO BE ; SET
      WAIT FOR ; IO READY
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;     32 DELAY
      EXECUTE A TOTAL OF ;      1 Z FROM P4 RECEIVED
      EXECUTE A TOTAL OF ;      1 Z ON P4 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; BROADCAST DX FROM P4
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
      CHAIN TO ; END BROADCAST Z
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
NAME = DX ON P1
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =       0.
   ALLOWED PROCESSORS =
      P1
   REQUIRED HARDWARE STATUS =
      P1
   ORED PREDECESSOR LIST =
      D0ABZ ON P1
      D1ABZ ON P1
      D2ABZ ON P1
      D3ABZ ON P1
      D4ABZ ON P1
      D5ABZ ON P1
      D6ABZ ON P1
      D7ABZ ON P1
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; Z FROM P2 RECEIVED
      TO BE ; SET
      WAIT FOR ; Z FROM P3 RECEIVED
      TO BE ; SET
      WAIT FOR ; Z FROM P4 RECEIVED
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;       1 DX ON P1
      EXECUTE A TOTAL OF ;       1 DX ON P1 READY
      EXECUTE A TOTAL OF ;       1 DX FROM P1 READY
NAME = DX ON P2
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =       0.
   ALLOWED PROCESSORS =
      P2
   REQUIRED HARDWARE STATUS =
      P2
   ORED PREDECESSOR LIST =
      D0ABZ ON P2
      D1ABZ ON P2
      D2ABZ ON P2
      D3ABZ ON P2
      D4ABZ ON P2
      D5ABZ ON P2
      D6ABZ ON P2
      D7ABZ ON P2
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; Z FROM P1 RECEIVED
      TO BE ; SET
      WAIT FOR ; Z FROM P3 RECEIVED
      TO BE ; SET
      WAIT FOR ; Z FROM P4 RECEIVED
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;       1 DX ON P2
      EXECUTE A TOTAL OF ;       1 DX ON P2 READY
      EXECUTE A TOTAL OF ;       1 DX FROM P2 READY
NAME = DX ON P3
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =       0.
   ALLOWED PROCESSORS =
      P3
   REQUIRED HARDWARE STATUS =
      P3
   ORED PREDECESSOR LIST =
      D0ABZ ON P3
      D1ABZ ON P3
```

```
         D2ABZ ON P3
         D3ABZ ON P3
         D4ABZ ON P3
         D5ABZ ON P3
         D6ABZ ON P3
         D7ABZ ON P3
      REQUIRED SEMAPHORE STATUS =
         WAIT FOR ; Z FROM P1 RECEIVED
         TO BE ; SET
         WAIT FOR ; Z FROM P2 RECEIVED
         TO BE ; SET
         WAIT FOR ; Z FROM P4 RECEIVED
         TO BE ; SET
      INSTRUCTION LIST =
         EXECUTE A TOTAL OF ;        1 DX ON P3
         EXECUTE A TOTAL OF ;        1 DX ON P3 READY
         EXECUTE A TOTAL OF ;        1 DX FROM P3 READY
   NAME = DX ON P4
      INTERRUPTABILITY FLAG = NO
      CONCURRENT EXECUTION = NO
      START TIME =       0.
      ALLOWED PROCESSORS =
         P4
      REQUIRED HARDWARE STATUS =
         P4
      ORED PREDECESSOR LIST =
         D0ABZ ON P4
         D1ABZ ON P4
         D2ABZ ON P4
         D3ABZ ON P4
         D4ABZ ON P4
         D5ABZ ON P4
         D6ABZ ON P4
         D7ABZ ON P4
      REQUIRED SEMAPHORE STATUS =
         WAIT FOR ; Z FROM P1 RECEIVED
         TO BE ; SET
         WAIT FOR ; Z FROM P2 RECEIVED
         TO BE ; SET
         WAIT FOR ; Z FROM P3 RECEIVED
         TO BE ; SET
      INSTRUCTION LIST =
         EXECUTE A TOTAL OF ;        1 DX ON P4
         EXECUTE A TOTAL OF ;        1 DX ON P4 READY
         EXECUTE A TOTAL OF ;        1 DX FROM P4 READY
   NAME = BROADCAST DX FROM P1
      INTERRUPTABILITY FLAG = NO
      CONCURRENT EXECUTION = NO
      START TIME =       0.
      ALLOWED PROCESSORS =
         P1IO
      REQUIRED HARDWARE STATUS =
         P1IO
      ANDED PREDECESSOR LIST =
         BROADCAST Z FROM P1
      REQUIRED SEMAPHORE STATUS =
         WAIT FOR ; DX FROM P1 READY
         TO BE ; SET
         WAIT FOR ; IO READY
         TO BE ; SET
      INSTRUCTION LIST =
         EXECUTE A TOTAL OF ;       32 DELAY
         EXECUTE A TOTAL OF ;        1 DX FROM P1 RECEIVED
         EXECUTE A TOTAL OF ;        1 DX FROM P1 NOT READY
      ANDED SUCCESSORS =
         BROADCAST Z FROM P1
         END BROADCAST DX
   NAME = BROADCAST X FROM P2
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =       0.
   ALLOWED PROCESSORS =
      P2IO
   REQUIRED HARDWARE STATUS =
      P2IO
   ANDED PREDECESSOR LIST =
      BROADCAST Z FROM P2
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; DX FROM P2 READY
      TO BE ; SET
      WAIT FOR ; IO READY
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;      32 DELAY
      EXECUTE A TOTAL OF ;       1 DX FROM P2 RECEIVED
      EXECUTE A TOTAL OF ;       1 DX FROM P2 NOT READY
   ANDED SUCCESSORS =
      BROADCAST Z FROM P2
      END BROADCAST DX
NAME = BROADCAST DX FROM P3
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =       0.
   ALLOWED PROCESSORS =
      P3IO
   REQUIRED HARDWARE STATUS =
      P3IO
   ANDED PREDECESSOR LIST =
      BROADCAST Z FROM P3
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; DX FROM P3 READY
      TO BE ; SET
      WAIT FOR ; IO READY
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;      32 DELAY
      EXECUTE A TOTAL OF ;       1 DX FROM P3 RECEIVED
      EXECUTE A TOTAL OF ;       1 DX FROM P3 NOT READY
   ANDED SUCCESSORS =
      BROADCAST Z FROM P3
      END BROADCAST DX
NAME = BROADCAST DX FROM P4
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =       0.
   ALLOWED PROCESSORS =
      P4IO
   REQUIRED HARDWARE STATUS =
      P4IO
   ANDED PREDECESSOR LIST =
      BROADCAST Z FROM P4
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; DX FROM P4 READY
      TO BE ; SET
      WAIT FOR ; IO READY
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;      32 DELAY
      EXECUTE A TOTAL OF ;       1 DX FROM P4 RECEIVED
      EXECUTE A TOTAL OF ;       1 FROM ON P4 NOT READY
   ANDED SUCCESSORS =
      BROADCAST Z FROM P4
      END BROADCAST DX
NAME = D1ABZ ON P1
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =       0.
   REQUIRED HARDWARE STATUS =
      P1
   ANDED PREDECESSOR LIST =
      D0ABZ ON P1
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; DX ON P1 READY
      TO BE ; SET
      WAIT FOR ; DX FROM P2 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P3 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P4 RECEIVED
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;       1 D1ABZ ON P1
      EXECUTE A TOTAL OF ;       1 Z ON P1 READY
      EXECUTE A TOTAL OF ;       1 DX ON P1 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; D2ABZ ON P1
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
      CHAIN TO ; DX ON P1
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
NAME = D1ABZ ON P2
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =       0.
   REQUIRED HARDWARE STATUS =
      P2
   ANDED PREDECESSOR LIST =
      D0ABZ ON P2
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; DX ON P2 READY
      TO BE ; SET
      WAIT FOR ; DX FROM P1 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P3 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P4 RECEIVED
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;       1 D1ABZ ON P2
      EXECUTE A TOTAL OF ;       1 Z ON P2 READY
      EXECUTE A TOTAL OF ;       1 DX ON P2 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; D2ABZ ON P2
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
      CHAIN TO ; DX ON P2
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
NAME = D1ABZ ON P3
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =       0.
   REQUIRED HARDWARE STATUS =
      P3
   ANDED PREDECESSOR LIST =
      D0ABZ ON P3
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; DX ON P3 READY
      TO BE ; SET
      WAIT FOR ; DX FROM P1 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P2 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P4 RECEIVED
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;       1 D1ABZ ON P3
      EXECUTE A TOTAL OF ;       1 Z ON P3 READY
      EXECUTE A TOTAL OF ;       1 DX ON P3 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; D2ABZ ON P3
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
      CHAIN TO ; DX ON P3
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
NAME = D1ABZ ON P4
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =      0.
   REQUIRED HARDWARE STATUS =
      P4
   ANDED PREDECESSOR LIST =
      D0ABZ ON P4
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; DX ON P4 READY
      TO BE ; SET
      WAIT FOR ; DX FROM P1 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P2 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P3 RECEIVED
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;       1 D1ABZ ON P4
      EXECUTE A TOTAL OF ;       1 Z ON P4 READY
      EXECUTE A TOTAL OF ;       1 DX ON P4 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; D2ABZ ON P4
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
      CHAIN TO ; DX ON P4
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
NAME = D2ABZ ON P1
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =      0.
   REQUIRED HARDWARE STATUS =
      P1
   ANDED PREDECESSOR LIST =
      D1ABZ ON P1
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; DX ON P1 READY
      TO BE ; SET
      WAIT FOR ; DX FROM P2 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P3 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P4 RECEIVED
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;       1 D2ABZ ON P1
      EXECUTE A TOTAL OF ;       1 Z ON P1 READY
      EXECUTE A TOTAL OF ;       1 DX ON P1 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; D3ABZ ON P1
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
      CHAIN TO ; DX ON P1
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
NAME = D2ABZ ON P2
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =      0.
   REQUIRED HARDWARE STATUS =
      P2
   ANDED PREDECESSOR LIST =
      D1ABZ ON P2
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; DX ON P2 READY
      TO BE ; SET
      WAIT FOR ; DX FROM P1 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P3 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P4 RECEIVED
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;      1 D2ABZ ON P2
      EXECUTE A TOTAL OF ;      1 Z ON P2 READY
      EXECUTE A TOTAL OF ;      1 DX ON P2 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; D3ABZ ON P2
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
      CHAIN TO ; DX ON P2
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
NAME = D2ABZ ON P3
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =      0.
   REQUIRED HARDWARE STATUS =
      P3
   ANDED PREDECESSOR LIST =
      D1ABZ ON P3
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; DX ON P3 READY
      TO BE ; SET
      WAIT FOR ; DX FROM P1 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P2 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P4 RECEIVED
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;      1 D2ABZ ON P3
      EXECUTE A TOTAL OF ;      1 Z ON P3 READY
      EXECUTE A TOTAL OF ;      1 DX ON P3 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; D3ABZ ON P3
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
      CHAIN TO ; DX ON P3
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
NAME = D2ABZ ON P4
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =      0.
   REQUIRED HARDWARE STATUS =
      P4
   ANDED PREDECESSOR LIST =
      D1ABZ ON P4
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; DX ON P4 READY
      TO BE ; SET
      WAIT FOR ; DX FROM P1 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P2 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P3 RECEIVED
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;      1 D2ABZ ON P4
      EXECUTE A TOTAL OF ;      1 Z ON P4 READY
      EXECUTE A TOTAL OF ;      1 DX ON P4 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; D3ABZ ON P4
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
      CHAIN TO ; DX ON P4
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
NAME = D3ABZ ON P1
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =      0.
   REQUIRED HARDWARE STATUS =
      P1
   ANDED PREDECESSOR LIST =
      D2ABZ ON P1
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; DX ON P1 READY
      TO BE ; SET
      WAIT FOR ; DX FROM P2 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P3 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P4 RECEIVED
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;      1 D3ABZ ON P1
      EXECUTE A TOTAL OF ;      1 Z ON P1 READY
      EXECUTE A TOTAL OF ;      1 DX ON P1 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; D4ABZ ON P1
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
      CHAIN TO ; DX ON P1
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
NAME = D3ABZ ON P2
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =      0.
   REQUIRED HARDWARE STATUS =
      P2
   ANDED PREDECESSOR LIST =
      D2ABZ ON P2
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; DX ON P2 READY
      TO BE ; SET
      WAIT FOR ; DX FROM P1 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P3 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P4 RECEIVED
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;      1 D3ABZ ON P2
      EXECUTE A TOTAL OF ;      1 Z ON P2 READY
      EXECUTE A TOTAL OF ;      1 DX ON P2 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; D4ABZ ON P2
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
      CHAIN TO ; DX ON P2
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
NAME = D3ABZ ON P3
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =      0.
   REQUIRED HARDWARE STATUS =
      P3
   ANDED PREDECESSOR LIST =
      D2ABZ ON P3
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; DX ON P3 READY
      TO BE ; SET
      WAIT FOR ; DX FROM P1 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P2 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P4 RECEIVED
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;      1 D3ABZ ON P3
      EXECUTE A TOTAL OF ;      1 Z ON P3 READY
      EXECUTE A TOTAL OF ;      1 DX ON P3 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; D4ABZ ON P3
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
      CHAIN TO ; DX ON P3
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
NAME = D3ABZ ON P4
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =      0.
   REQUIRED HARDWARE STATUS =
      P4
   ANDED PREDECESSOR LIST =
      D2ABZ ON P4
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; DX ON P4 READY
      TO BE ; SET
      WAIT FOR ; DX FROM P1 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P2 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P3 RECEIVED
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;      1 D3ABZ ON P4
      EXECUTE A TOTAL OF ;      1 Z ON P4 READY
      EXECUTE A TOTAL OF ;      1 DX ON P4 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; D4ABZ ON P4
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
      CHAIN TO ; DX ON P4
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
NAME = D4ABZ ON P1
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =      0.
   REQUIRED HARDWARE STATUS =
      P1
   ANDED PREDECESSOR LIST =
      D3ABZ ON P1
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; DX ON P1 READY
      TO BE ; SET
      WAIT FOR ; DX FROM P2 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P3 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P4 RECEIVED
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;      1 D4ABZ ON P1
      EXECUTE A TOTAL OF ;      1 Z ON P1 READY
      EXECUTE A TOTAL OF ;      1 DX ON P1 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; D5ABZ ON P1
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
      CHAIN TO ; DX ON P1
      WITH ITERATIONS THEN SKIP COUNT OF ;      0
NAME = D4ABZ ON P2
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =      0.
   REQUIRED HARDWARE STATUS =
         P2
      ANDED PREDECESSOR LIST =
         D3ABZ ON P2
      REQUIRED SEMAPHORE STATUS =
         WAIT FOR ; DX ON P2 READY
         TO BE ; SET
         WAIT FOR ; DX FROM P1 RECEIVED
         TO BE ; SET
         WAIT FOR ; DX FROM P3 RECEIVED
         TO BE ; SET
         WAIT FOR ; DX FROM P4 RECEIVED
         TO BE ; SET
      INSTRUCTION LIST =
         EXECUTE A TOTAL OF ;       1 D4ABZ ON P2
         EXECUTE A TOTAL OF ;       1 Z ON P2 READY
         EXECUTE A TOTAL OF ;       1 DX ON P2 NOT READY
      ANDED SUCCESSORS =
         CHAIN TO ; D5ABZ ON P2
         WITH ITERATIONS THEN SKIP COUNT OF ;       0
         CHAIN TO ; DX ON P2
         WITH ITERATIONS THEN SKIP COUNT OF ;       0
   NAME = D4ABZ ON P3
      INTERRUPTABILITY FLAG = NO
      CONCURRENT EXECUTION = NO
      START TIME =      0.
      REQUIRED HARDWARE STATUS =
         P3
      ANDED PREDECESSOR LIST =
         D3ABZ ON P3
      REQUIRED SEMAPHORE STATUS =
         WAIT FOR ; DX ON P3 READY
         TO BE ; SET
         WAIT FOR ; DX FROM P1 RECEIVED
         TO BE ; SET
         WAIT FOR ; DX FROM P2 RECEIVED
         TO BE ; SET
         WAIT FOR ; DX FROM P4 RECEIVED
         TO BE ; SET
      INSTRUCTION LIST =
         EXECUTE A TOTAL OF ;       1 D4ABZ ON P3
         EXECUTE A TOTAL OF ;       1 Z ON P3 READY
         EXECUTE A TOTAL OF ;       1 DX ON P3 NOT READY
      ANDED SUCCESSORS =
         CHAIN TO ; D5ABZ ON P3
         WITH ITERATIONS THEN SKIP COUNT OF ;       0
         CHAIN TO ; DX ON P3
         WITH ITERATIONS THEN SKIP COUNT OF ;       0
   NAME = D4ABZ ON P4
      INTERRUPTABILITY FLAG = NO
      CONCURRENT EXECUTION = NO
      START TIME =      0.
      REQUIRED HARDWARE STATUS =
         P4
      ANDED PREDECESSOR LIST =
         D3ABZ ON P4
      REQUIRED SEMAPHORE STATUS =
         WAIT FOR ; DX ON P4 READY
         TO BE ; SET
         WAIT FOR ; DX FROM P1 RECEIVED
         TO BE ; SET
         WAIT FOR ; DX FROM P2 RECEIVED
         TO BE ; SET
         WAIT FOR ; DX FROM P3 RECEIVED
         TO BE ; SET
      INSTRUCTION LIST =
         EXECUTE A TOTAL OF ;       1 D4ABZ ON P4
         EXECUTE A TOTAL OF ;       1 Z ON P4 READY
         EXECUTE A TOTAL OF ;       1 DX ON P4 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; D5ABZ ON P4
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
      CHAIN TO ; DX ON P4
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
NAME = D5ABZ ON P1
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =      0.
   REQUIRED HARDWARE STATUS =
      P1
   ANDED PREDECESSOR LIST =
      D4ABZ ON P1
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; DX ON P1 READY
      TO BE ; SET
      WAIT FOR ; DX FROM P2 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P3 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P4 RECEIVED
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;      1 D5ABZ ON P1
      EXECUTE A TOTAL OF ;      1 Z ON P1 READY
      EXECUTE A TOTAL OF ;      1 DX ON P1 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; D6ABZ ON P1
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
      CHAIN TO ; DX ON P1
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
NAME = D5ABZ ON P2
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =      0.
   REQUIRED HARDWARE STATUS =
      P2
   ANDED PREDECESSOR LIST =
      D4ABZ ON P2
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; DX ON P2 READY
      TO BE ; SET
      WAIT FOR ; DX FROM P1 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P3 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P4 RECEIVED
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;      1 D5ABZ ON P2
      EXECUTE A TOTAL OF ;      1 Z ON P2 READY
      EXECUTE A TOTAL OF ;      1 DX ON P2 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; D6ABZ ON P2
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
      CHAIN TO ; DX ON P2
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
NAME = D5ABZ ON P3
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =      0.
   REQUIRED HARDWARE STATUS =
      P3
   ANDED PREDECESSOR LIST =
      D4ABZ ON P3
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; DX ON P3 READY
      TO BE ; SET
      WAIT FOR ; DX FROM P1 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P2 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P4 RECEIVED
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;       1 D5ABZ ON P3
      EXECUTE A TOTAL OF ;       1 Z ON P3 READY
      EXECUTE A TOTAL OF ;       1 DX ON P3 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; D6ABZ ON P3
      WITH ITERATIONS THEN SKIP COUNT OF ;        0
      CHAIN TO ; DX ON P3
      WITH ITERATIONS THEN SKIP COUNT OF ;        0
NAME = D5ABZ ON P4
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =       0.
   REQUIRED HARDWARE STATUS =
      P4
   ANDED PREDECESSOR LIST =
      D4ABZ ON P4
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; DX ON P4 READY
      TO BE ; SET
      WAIT FOR ; DX FROM P1 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P2 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P3 RECEIVED
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;       1 D5ABZ ON P4
      EXECUTE A TOTAL OF ;       1 Z ON P4 READY
      EXECUTE A TOTAL OF ;       1 DX ON P4 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; D6ABZ ON P4
      WITH ITERATIONS THEN SKIP COUNT OF ;        0
      CHAIN TO ; DX ON P4
      WITH ITERATIONS THEN SKIP COUNT OF ;        0
NAME = D6ABZ ON P1
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =       0.
   REQUIRED HARDWARE STATUS =
      P1
   ANDED PREDECESSOR LIST =
      D5ABZ ON P1
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; DX ON P1 READY
      TO BE ; SET
      WAIT FOR ; DX FROM P2 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P3 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P4 RECEIVED
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;       1 D6ABZ ON P1
      EXECUTE A TOTAL OF ;       1 Z ON P1 READY
      EXECUTE A TOTAL OF ;       1 DX ON P1 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; D7ABZ ON P1
      WITH ITERATIONS THEN SKIP COUNT OF ;        0
      CHAIN TO ; DX ON P1
      WITH ITERATIONS THEN SKIP COUNT OF ;        0
NAME = D6ABZ ON P2
```

```
      INTERRUPTABILITY FLAG = NO
      CONCURRENT EXECUTION = NO
      START TIME =       0.
      REQUIRED HARDWARE STATUS =
         P2
      ANDED PREDECESSOR LIST =
         D5ABZ ON P2
      REQUIRED SEMAPHORE STATUS =
         WAIT FOR ; DX ON P2 READY
         TO BE ; SET
         WAIT FOR ; DX FROM P1 RECEIVED
         TO BE ; SET
         WAIT FOR ; DX FROM P3 RECEIVED
         TO BE ; SET
         WAIT FOR ; DX FROM P4 RECEIVED
         TO BE ; SET
      INSTRUCTION LIST =
         EXECUTE A TOTAL OF ;      1 D6ABZ ON P2
         EXECUTE A TOTAL OF ;      1 Z ON P2 READY
         EXECUTE A TOTAL OF ;      1 DX ON P2 NOT READY
      ANDED SUCCESSORS =
         CHAIN TO ; D7ABZ ON P2
         WITH ITERATIONS THEN SKIP COUNT OF ;      0
         CHAIN TO ; DX ON P2
         WITH ITERATIONS THEN SKIP COUNT OF ;      0
   NAME = D6ABZ ON P3
      INTERRUPTABILITY FLAG = NO
      CONCURRENT EXECUTION = NO
      START TIME =       0.
      REQUIRED HARDWARE STATUS =
         P3
      ANDED PREDECESSOR LIST =
         D5ABZ ON P3
      REQUIRED SEMAPHORE STATUS =
         WAIT FOR ; DX ON P3 READY
         TO BE ; SET
         WAIT FOR ; DX FROM P1 RECEIVED
         TO BE ; SET
         WAIT FOR ; DX FROM P2 RECEIVED
         TO BE ; SET
         WAIT FOR ; DX FROM P4 RECEIVED
         TO BE ; SET
      INSTRUCTION LIST =
         EXECUTE A TOTAL OF ;      1 D6ABZ ON P3
         EXECUTE A TOTAL OF ;      1 Z ON P3 READY
         EXECUTE A TOTAL OF ;      1 DX ON P3 NOT READY
      ANDED SUCCESSORS =
         CHAIN TO ; D7ABZ ON P3
         WITH ITERATIONS THEN SKIP COUNT OF ;      0
         CHAIN TO ; DX ON P3
         WITH ITERATIONS THEN SKIP COUNT OF ;      0
   NAME = D6ABZ ON P4
      INTERRUPTABILITY FLAG = NO
      CONCURRENT EXECUTION = NO
      START TIME =       0.
      REQUIRED HARDWARE STATUS =
         P4
      ANDED PREDECESSOR LIST =
         D5ABZ ON P4
      REQUIRED SEMAPHORE STATUS =
         WAIT FOR ; DX ON P4 READY
         TO BE ; SET
         WAIT FOR ; DX FROM P1 RECEIVED
         TO BE ; SET
         WAIT FOR ; DX FROM P2 RECEIVED
         TO BE ; SET
         WAIT FOR ; DX FROM P3 RECEIVED
         TO BE ; SET
      INSTRUCTION LIST =
         EXECUTE A TOTAL OF ;       1 D6ABZ ON P4
         EXECUTE A TOTAL OF ;       1 Z ON P4 READY
         EXECUTE A TOTAL OF ;       1 DX ON P4 NOT READY
      ANDED SUCCESSORS =
         CHAIN TO ; D7ABZ ON P4
         WITH ITERATIONS THEN SKIP COUNT OF ;       0
         CHAIN TO ; DX ON P4
         WITH ITERATIONS THEN SKIP COUNT OF ;       0
   NAME = D7ABZ ON P1
      INTERRUPTABILITY FLAG = NO
      CONCURRENT EXECUTION = NO
      START TIME =      0.
      REQUIRED HARDWARE STATUS =
         P1
      ANDED PREDECESSOR LIST =
         D6ABZ ON P1
      REQUIRED SEMAPHORE STATUS =
         WAIT FOR ; DX ON P1 READY
         TO BE ; SET
         WAIT FOR ; DX FROM P2 RECEIVED
         TO BE ; SET
         WAIT FOR ; DX FROM P3 RECEIVED
         TO BE ; SET
         WAIT FOR ; DX FROM P4 RECEIVED
         TO BE ; SET
      INSTRUCTION LIST =
         EXECUTE A TOTAL OF ;       1 D7ABZ ON P1
         EXECUTE A TOTAL OF ;       1 Z ON P1 READY
         EXECUTE A TOTAL OF ;       1 DX ON P1 NOT READY
      ANDED SUCCESSORS =
         CHAIN TO ; D8ABZ ON P1
         WITH ITERATIONS THEN SKIP COUNT OF ;       0
         CHAIN TO ; DX ON P1
         WITH ITERATIONS THEN SKIP COUNT OF ;       0
   NAME = D7ABZ ON P2
      INTERRUPTABILITY FLAG = NO
      CONCURRENT EXECUTION = NO
      START TIME =      0.
      REQUIRED HARDWARE STATUS =
         P2
      ANDED PREDECESSOR LIST =
         D6ABZ ON P2
      REQUIRED SEMAPHORE STATUS =
         WAIT FOR ; DX ON P2 READY
         TO BE ; SET
         WAIT FOR ; DX FROM P1 RECEIVED
         TO BE ; SET
         WAIT FOR ; DX FROM P3 RECEIVED
         TO BE ; SET
         WAIT FOR ; DX FROM P4 RECEIVED
         TO BE ; SET
      INSTRUCTION LIST =
         EXECUTE A TOTAL OF ;       1 D7ABZ ON P2
         EXECUTE A TOTAL OF ;       1 Z ON P2 READY
         EXECUTE A TOTAL OF ;       1 DX ON P2 NOT READY
      ANDED SUCCESSORS =
         CHAIN TO ; D8ABZ ON P2
         WITH ITERATIONS THEN SKIP COUNT OF ;       0
         CHAIN TO ; DX ON P2
         WITH ITERATIONS THEN SKIP COUNT OF ;       0
   NAME = D7ABZ ON P3
      INTERRUPTABILITY FLAG = NO
      CONCURRENT EXECUTION = NO
      START TIME =      0.
      REQUIRED HARDWARE STATUS =
         P3
      ANDED PREDECESSOR LIST =
      D6ABZ ON P3
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; DX ON P3 READY
      TO BE ; SET
      WAIT FOR ; DX FROM P1 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P2 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P4 RECEIVED
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;       1 D7ABZ ON P3
      EXECUTE A TOTAL OF ;       1 Z ON P3 READY
      EXECUTE A TOTAL OF ;       1 DX ON P3 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; D8ABZ ON P3
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
      CHAIN TO ; DX ON P3
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
NAME = D7ABZ ON P4
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =       0.
   REQUIRED HARDWARE STATUS =
      P4
   ANDED PREDECESSOR LIST =
      D6ABZ ON P4
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; DX ON P4 READY
      TO BE ; SET
      WAIT FOR ; DX FROM P1 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P2 RECEIVED
      TO BE ; SET
      WAIT FOR ; DX FROM P3 RECEIVED
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;       1 D7ABZ ON P4
      EXECUTE A TOTAL OF ;       1 Z ON P4 READY
      EXECUTE A TOTAL OF ;       1 DX ON P4 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; D8ABZ ON P4
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
      CHAIN TO ; DX ON P4
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
NAME = NEW X ON P1
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =       0.
   ALLOWED PROCESSORS =
      P1
   REQUIRED HARDWARE STATUS =
      P1
   ANDED PREDECESSOR LIST =
      D7ABZ ON P1
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; DX ON P1 READY
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;      11 SERIES EVALUATION
      EXECUTE A TOTAL OF ;       1 P1 FINISHED
      EXECUTE A TOTAL OF ;       1 NEW X ON P1 READY
      EXECUTE A TOTAL OF ;       1 NEW X FROM P1 READY
   ANDED SUCCESSORS =
      CHAIN TO ; D0ABZ ON P1
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
NAME = NEW X ON P2
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =      0.
   ALLOWED PROCESSORS =
      P2
   REQUIRED HARDWARE STATUS =
      P2
   ANDED PREDECESSOR LIST =
      D7ABZ ON P2
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; DX ON P2 READY
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;      14 SERIES EVALUATION
      EXECUTE A TOTAL OF ;       1 P2 FINISHED
      EXECUTE A TOTAL OF ;       1 NEW X ON P2 READY
      EXECUTE A TOTAL OF ;       1 NEW X FROM P2 READY
   ANDED SUCCESSORS =
      CHAIN TO ; D0ABZ ON P2
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
NAME = NEW X ON P3
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =      0.
   ALLOWED PROCESSORS =
      P3
   REQUIRED HARDWARE STATUS =
      P3
   ANDED PREDECESSOR LIST =
      D7ABZ ON P3
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; DX ON P3 READY
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;      10 SERIES EVALUATION
      EXECUTE A TOTAL OF ;       1 P3 FINISHED
      EXECUTE A TOTAL OF ;       1 NEW X ON P3 READY
      EXECUTE A TOTAL OF ;       1 NEW X FROM P3 READY
   ANDED SUCCESSORS =
      CHAIN TO ; D0ABZ ON P3
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
NAME = NEW X ON P4
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =      0.
   ALLOWED PROCESSORS =
      P4
   REQUIRED HARDWARE STATUS =
      P4
   ANDED PREDECESSOR LIST =
      D7ABZ ON P4
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; DX ON P4 READY
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;      13 SERIES EVALUATION
      EXECUTE A TOTAL OF ;       1 P4 FINISHED
      EXECUTE A TOTAL OF ;       1 NEW X ON P3 READY
      EXECUTE A TOTAL OF ;       1 NEW X FROM P3 READY
   ANDED SUCCESSORS =
      CHAIN TO ; D0ABZ ON P4
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
NAME = BROADCAST NEW X FROM P1
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =      0.
   ALLOWED PROCESSORS =
      P1IO
   REQUIRED HARDWARE STATUS =
            P1IO
         ORED PREDECESSOR LIST =
            INITIALIZATION OF P1IO
            BROADCAST NEW X FROM P1
         REQUIRED SEMAPHORE STATUS =
            WAIT FOR ; NEW X FROM P1 READY
            TO BE ; SET
            WAIT FOR ; IO READY
            TO BE ; SET
         INSTRUCTION LIST =
            EXECUTE A TOTAL OF ;      32 DELAY
            EXECUTE A TOTAL OF ;       1 NEW X FROM P1 RECEIVED
            EXECUTE A TOTAL OF ;       1 NEW X FROM P1 NOT READY
         ANDED SUCCESSORS =
            CHAIN TO ; BROADCAST NEW X FROM P1
            WITH ITERATIONS THEN SKIP COUNT OF ;       0
            CHAIN TO ; END BROADCAST NEW X
            WITH ITERATIONS THEN SKIP COUNT OF ;       0
      NAME = BROADCAST NEW X FROM P2
         INTERRUPTABILITY FLAG = NO
         CONCURRENT EXECUTION = NO
         START TIME =       0.
         ALLOWED PROCESSORS =
            P2IO
         REQUIRED HARDWARE STATUS =
            P2IO
         ORED PREDECESSOR LIST =
            INITIALIZATION OF P1IO
            BROADCAST NEW X FROM P2
         REQUIRED SEMAPHORE STATUS =
            WAIT FOR ; NEW X FROM P2 READY
            TO BE ; SET
            WAIT FOR ; IO READY
            TO BE ; SET
         INSTRUCTION LIST =
            EXECUTE A TOTAL OF ;      32 DELAY
            EXECUTE A TOTAL OF ;       1 NEW X FROM P2 RECEIVED
            EXECUTE A TOTAL OF ;       1 NEW X FROM P2 NOT READY
         ANDED SUCCESSORS =
            CHAIN TO ; BROADCAST NEW X FROM P2
            WITH ITERATIONS THEN SKIP COUNT OF ;       0
            CHAIN TO ; END BROADCAST NEW X
            WITH ITERATIONS THEN SKIP COUNT OF ;       0
      NAME = BROADCAST NEW X FROM P3
         INTERRUPTABILITY FLAG = NO
         CONCURRENT EXECUTION = NO
         START TIME =       0.
         ALLOWED PROCESSORS =
            P3IO
         REQUIRED HARDWARE STATUS =
            P3IO
         ORED PREDECESSOR LIST =
            INITIALIZATION OF P3IO
            BROADCAST NEW X FROM P3
         REQUIRED SEMAPHORE STATUS =
            WAIT FOR ; NEW X FROM P3 READY
            TO BE ; SET
            WAIT FOR ; IO READY
            TO BE ; SET
         INSTRUCTION LIST =
            EXECUTE A TOTAL OF ;      32 DELAY
            EXECUTE A TOTAL OF ;       1 NEW X FROM P3 RECEIVED
            EXECUTE A TOTAL OF ;       1 NEW X FROM P3 NOT READY
         ANDED SUCCESSORS =
            CHAIN TO ; BROADCAST NEW X FROM P3
            WITH ITERATIONS THEN SKIP COUNT OF ;       0
            CHAIN TO ; END BROADCAST NEW X
            WITH ITERATIONS THEN SKIP COUNT OF ;       0
```

```
NAME = BROADCAST NEW X FROM P4
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =       0.
   ALLOWED PROCESSORS =
      P4IO
   REQUIRED HARDWARE STATUS =
      P4IO
   ORED PREDECESSOR LIST =
      INITIALIZATION OF P4IO
      BROADCAST NEW X FROM P4
   REQUIRED SEMAPHORE STATUS =
      WAIT FOR ; NEW X FROM P4 READY
      TO BE ; SET
      WAIT FOR ; IO READY
      TO BE ; SET
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;      32 DELAY
      EXECUTE A TOTAL OF ;       1 NEW X FROM P4 RECEIVED
      EXECUTE A TOTAL OF ;       1 NEW X FROM P4 NOT READY
   ANDED SUCCESSORS =
      CHAIN TO ; BROADCAST NEW X FROM P4
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
      CHAIN TO ; END BROADCAST NEW X
      WITH ITERATIONS THEN SKIP COUNT OF ;       0
NAME = END BROADCAST NEW X
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =       0.
   ALLOWED PROCESSORS =
      P1IO
   REQUIRED HARDWARE STATUS =
      P1IO
   ANDED PREDECESSOR LIST =
      BROADCAST NEW X FROM P1
      BROADCAST NEW X FROM P2
      BROADCAST NEW X FROM P3
      BROADCAST NEW X FROM P4
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;       1 NEW X FROM P1 NOT RECEIVED
      EXECUTE A TOTAL OF ;       1 NEW X FROM P2 NOT RECEIVED
      EXECUTE A TOTAL OF ;       1 NEW X FROM P3 NOT RECEIVED
      EXECUTE A TOTAL OF ;       1 NEW X FROM P4 NOT RECEIVED
NAME = END BROADCAST DX
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =       0.
   ALLOWED PROCESSORS =
      P1IO
   REQUIRED HARDWARE STATUS =
      P1IO
   ANDED PREDECESSOR LIST =
      BROADCAST DX FROM P1
      BROADCAST DX FROM P2
      BROADCAST DX FROM P3
      BROADCAST DX FROM P4
   INSTRUCTION LIST =
      EXECUTE A TOTAL OF ;       1 DX FROM P1 NOT RECEIVED
      EXECUTE A TOTAL OF ;       1 DX FROM P2 NOT RECEIVED
      EXECUTE A TOTAL OF ;       1 DX FROM P3 NOT RECEIVED
      EXECUTE A TOTAL OF ;       1 DX FROM P4 NOT RECEIVED
NAME = END BROADCAST Z
   INTERRUPTABILITY FLAG = NO
   CONCURRENT EXECUTION = NO
   START TIME =       0.
   ALLOWED PROCESSORS =
      P1IO
   REQUIRED HARDWARE STATUS =
   P1IO
ANDED PREDECESSOR LIST =
   BROADCAST Z FROM P1
   BROADCAST Z FROM P2
   BROADCAST Z FROM P3
INSTRUCTION LIST =
   EXECUTE A TOTAL OF ;       1 Z FROM P1 NOT RECEIVED
   EXECUTE A TOTAL OF ;       1 Z FROM P2 NOT RECEIVED
   EXECUTE A TOTAL OF ;       1 Z FROM P3 NOT RECEIVED
   EXECUTE A TOTAL OF ;       1 Z FROM P4 NOT RECEIVED
```

APPENDIX B

The benchmark ODE systems

As explained in Chapter 5, the performance analysis of the procedures for the selection of algorithms that solve ODE systems and the multiprocessor implementation of these algorithms requires the experimental application of these procedures on a set of benchmark ODE systems.

Chapter 6 suggests that the following three benchmarks are required for the analysis of the algorithm selection procedure:

benchmark #1

This ODE system is characterized by:

1. 50% sparsity for the matrix A distributed linearly over its rows from 42% to 58%.
2. 90% sparsity for the matrices $\alpha, \beta, \gamma, \delta$ and θ of each nonzero element of A.
3. Linear terms as large as the nonlinear terms in the nonzero elements of A.
4. 50 Hz highest frequency.

```
A(1,1)=1.263+4.085*x8*cosx10
A(1,3)=2.839-0.916*x6-1.211*x1*x9+2.082*x1*x11-2.248*x9*
       x5
A(1,7)=3.421-2.195*x2-3.39*x4*x4+0.121*x4*x10-1.124*x10*
       x2*cosx1+4.634*x10*x2*cosx7+0.159*x10*x10*cosx2+
       0.355*x10*x10*cosx4-2.437*x10*x10*cosx10+4.2*x10*
       x12*cosx6+0.951*x10*x12*cosx9
```

```
A(1,8)=3.318+0.003*x3*x9-3.12*x8*x5+1.617*x8*x8+3.048*
       x3*x9*cosx1-4.993*x3*x9*cosx2+1.141*x4*x11*cosx7+
       2.838*x7*x9*cosx6+2.779*x7*x12*cosx8
A(2,6)=0.619-0.207*x3-1.030*x9-1.275*x3*x1+1.674*x3*x4-
       0.147*x6*x5-1.132*x6*x8
A(2,7)=3.129+2.676*x3+2.64*x4+1.192*x1*x10-4.122*x1*cosx8
A(2,8)=3.192-4.771*x8-1.867*x1*x1-2.621*x6*x4+2.013*x6*
       x11+1.313*x5*cosx2-0.086*x5*cosx7+2.089*x8*cosx6-
       3.585*x8*x6*cosx3+3.133*x8*x6*cosx9
A(3,3)=1.582+2.729*x1-2.173*x2*cosx11-4.044*x2*cosx12-
       0.48*x6*x3*cosx3-4.934*x6*x10*cosx6
A(3,4)=1.814-4.849*x3-1.527*x11+1.799*x3*x2-1.218*x2*
       cosx11
A(3,7)=0.843+4.323*x2+0.474*x6+2.74*x1*x7+4.852*x3*x6-
       2.375*x2*cosx11
A(3,9)=2.287+4.289*x11*x2*cosx10-2.089*x11*x2*cosx12
A(3,10)=2.2-4.438*x2+3.007*x10+2.222*x1*x1+3.044*x1*x7-
        2.932*x3*cosx10+4.033*x3*cosx12
A(3,12)=2.659-3.492*x3+4.241*x10-2.138*x5*x4-2.174*x10*
        x9+2.785*x2*cosx4-4.509*x2*cosx7-4.579*x2*cosx10-
        4.193*x6*cosx10
A(4,1)=4.578-0.954*x3*x2*cosx11+3.576*x3*x5*cosx8-2.38*
       x3*x11*cosx7-4.374*x3*x11*cosx9-3.627*x3*x11*
       cosx12-4.411*x3*x12*cosx1-2.132*x3*x12*cosx5
A(4,3)=2.648+3.067*x7
A(4,5)=3.304+1.131*x6+2.5*x11*x1*cosx3-1.935*x11*x5*
       cosx2-1.621*x11*x5*cosx5-0.4*x11*x5*cosx7-1.026*
       x11*x5*cosx12
A(4,7)=2.928+4.542*x9*cosx5-2.742*x9*cosx7+
A(4,9)=4.724-0.105*x5-3.296*x10+0.547*x5*x8*cosx6+2.642*
       x5*x8*cosx8-2.451*x8*x11*cosx8
A(4,11)=2.261-2.644*x1*x4+2.765*x9*x6
A(5,4)=1.375-2.249*x8-3.538*x9+0.949*x2*cosx6+4.454*x1*
       x1*cosx9-3.794*x1*x1*cosx11+3.172*x1*x10*cosx7+
       1.694*x2*x6*cosx1+4.72*x2*x6*cosx3+4.334*x6*x4*
       cosx10+1.782*x6*x4*cosx12
A(5,5)=2.191+3.091*x10+2.008*x10*x2+0.292*x6*cosx9-4.489*
       x1*x6*cosx11-1.898*x1*x6*cosx12
A(5,7)=4.905-0.552*x11-1.422*x5*x4+3.355*x5*x11+1.073*x9*
       x12*cosx9
A(5,8)=4.896-3.706*x11-2.876*x1*x4*cosx2+0.457*x1*x4*
       cosx10+1.999*x1*x12*cosx11
A(5,10)=3.137+3.621*x10-1.608*x7*x6-4.298*x7*x7+0.106*x4*
        cosx1+1.216*x9*cosx4-1.484*x9*cosx10+3.01*x5*x4*
        cosx3+2.382*x5*x4*cosx7
A(5,11)=2.974-3.948*x2-3.162*x6-1.439*x11+4.533*x3*x4-
        1.551*x3*x5+3.723*x3*x8
A(5,12)=1.484+4.21*x6+4.542*x1*x2-3.625*x1*x7+2.92*x12*
        cosx1+0.979*x12*cosx8
A(6,1)=2.38+3.096*x2-2.901*x11-2.654*x1*cosx3-0.067*x11*
       cosx11-4.153*x12*cosx12
A(6,2)=1.316+4.106*x1-4.263*x2-4.01*x6-4.795*x8
A(6,5)=4.844-2.198*x5+3.613*x4*x11+0.059*x5*x1+1.062*x5*
       x8-2.354*x9*cosx8-3.839*x10*cosx5+3.284*x10*cosx10
A(6,6)=3.381-2.999*x3+2.207*x1*cosx2+4.064*x1*cosx7-3.002*
       x1*cosx8+0.933*x9*cosx8+1.388*x9*cosx10
A(6,7)=0.333-4.8*x6*cosx9+0.502*x6*cosx11
A(6,8)=0.501+4.523*x12*x11+0.599*x5*x5*cosx9+3.597*x5*x5*
       cosx11-3.203*x5*x9*cosx7+3.016*x11*x5*cosx12-
       1.839*x11*x12*cosx2-2.023*x11*x12*cosx3+0.428*x11*
       x12*cosx7
A(6,9)=1.206-2.839*x4*x3-4.841*x7*x4*cosx10+1.587*x7*x4*
       cosx12
A(6,10)=2.408-1.853*x2+1.314*x3-4.659*x1*x8+2.769*x2*
        cosx3+2.24*x1*x3*cosx11
A(6,12)=0.187+2.545*x6-2.015*x10+2.325*x4*x3*cosx9+3.3*
        x4*x3*cosx12
A(7,1)=2.028+3.847*x11*x4-3.68*x11*x10-0.788*x5*x4*
```

```
       cosx1-3.327*x5*x12*cosx1+1.794*x5*x12*cosx8
A(7,4)=4.218-0.794*x1*cosx1+4.689*x1*cosx10-1.105*x2*
       cosx3-2.536*x2*cosx7+0.054*x11*cosx6+0.937*x3*x5*
       cosx2+2.916*x3*x7*cosx1+1.275*x3*x7*cosx11-3.145*
       x7*x3*cosx3-1.256*x8*x12*cosx9+1.624*x10*x7*cosx11
A(7,5)=3.285+1.057*x4-1.699*x1*x4-4.7*x9*x8-0.421*x1*x1*
       cosx4
A(7,9)=0.746+0.766*x4-0.531*x6+2.335*x9*x2+1.129*x8*x6*
       cosx7-1.026*x8*x6*cosx10
A(8,1)=1.363+4.514*x2
A(8,4)=4.856-3.234*x7-1.157*x4*x2+4.886*x12*cosx6-1.143*
       x12*cosx12
A(8,5)=0.658+0.013*x3*x2-2.112*x3*x5-4.419*x3*x8
A(8,6)=4.340-1.775*x9+3.247*x2*cosx1+2.354*x5*cosx4+
       4.087*x5*cosx5+1.625*x1*x6*cosx2
A(8,7)=4.765-0.669*x5*x5+1.316*x5*x8-2.196*x7*cosx3-
       1.237*x7*cosx6
A(8,9)=0.402-4.816*x5*cosx2+1.338*x5*cosx4-1.341*x5*cosx5
A(8,11)=3.267+1.663*x11-2.614*x11*x8+2.996*x11*x11-0.77*
        x7*x4*cosx9-4.095*x7*x12*cosx4+0.86*x7*x12*
        cosx12+1.083*x11*x9*cosx4
A(8,12)=4.425-0.316*x6*x6*cosx11
A(9,1)=2.029+2.545*x7*cosx2-4.479*x7*cosx10-0.242*x9*
       cosx8+0.432*x9*cosx11
A(9,2)=4.034+4.501*x10+2.915*x3*x3+2.749*x3*x6-0.336*x3*
       x10-0.432*x5*x1
A(9,3)=1.757-1.699*x5+3.31*x1*x11+2.536*x5*x6+2.815*x5*
       x11+4.39*x6*x7+4.155*x4*cosx1+2.393*x4*cosx6+
       1.036*x4*cosx8-4.526*x10*cosx5+3.44*x10*cosx11-
       0.145*x5*x6*cosx2
A(9,6)=0.08+4.025*x2-3.608*x6-0.469*x10*x2+3.442*x12*x9+
       0.885*x1*cosx3-1.239*x9*cosx5
A(9,7)=4.585-1.023*x3+0.946*x5*x3*cosx3-2.261*x5*x3*
       cosx8+0.719*x5*x5*cosx6-2.713*x5*x5*cosx11-0.451*
       x5*x8*cosx6-0.209*x5*x8*cosx9
A(9,8)=3.831-4.429*x3+1.901*x2*x2-3.242*x2*x8+3.685*x7*
       cosx1
A(9,10)=4.307
A(10,2)=4.87+3.939*x4-1.104*x6+4.442*x11+1.052*x2*x3+
        4.111*x2*x4-2.406*x2*x5+0.686*x5*x6*cosx1-0.423*
        x5*x6*cosx7-4.826*x5*x8*cosx1-3.67*x5*x8*cosx2
A(10,3)=4.19+1.751*x1-4.157*x9-3.008*x1*x6-1.028*x5*
        cosx5-0.985*x12*x1*cosx4+4.156*x12*x2*cosx8+
        3.692*x12*x2*cosx11-1.439*x12*x2*cosx12-2.557*
        x12*x3*cosx12-4.437*x12*x12*cosx11+
A(10,5)=0.542-2.478*x2+0.548*x10-2.257*x11*cosx7-2.629*
        x3*x8*cosx12
A(10,6)=3.622+2.493*x2+0.879*x9-3.779*x2*x4+0.39*x2*x10+
        1.97*x12*x11-3.886*x2*cosx7-2.411*x2*cosx11-
        0.273*x2*cosx12+4.315*x10*x10*cosx5-1.897*x12*
        x4*cosx3+1.792*x12*x4*cosx7-4.947*x12*x10*cosx3-
        3.579*x12*x10*cosx9+4.866*x12*x10*cosx10
A(10,7)=4.745
A(10,9)=3.227-0.558*x8-4.465*x11+3.403*x6*cosx3+2.376*
        x10*cosx7+2.66*x4*x8*cosx12+
A(10,11)=2.385+4.147*x5-3.895*x12-0.134*x1*cosx5-2.872*
         x1*cosx10-0.677*x1*cosx12+2.169*x5*cosx6
A(10,12)=3.085-2.021*x8+2.2*x1*x8*cosx7-0.552*x1*x8*
         cosx11
A(11,2)=2.624-0.673*x2-1.818*x4+4.06*x4*cosx8-2.807*x9*
        cosx7-4.203*x9*cosx10-0.142*x3*x10*cosx12
A(11,6)=2.228+1.606*x1+2.043*x2-0.168*x7+4.572*x10+4.892*
        x12-3.988*x6*cosx2+2.243*x9*cosx10+2.193*x12*x5*
        cosx4+3.616*x12*x5*cosx6
A(11,7)=3.355+2.741*x1-0.824*x4+4.642*x3*x6+4.292*x5*x2+
        0.843*x5*x4+3.607*x9*cosx9-0.964*x3*x8*cosx10
A(11,8)=1.349-4.058*x3*cosx8+1.153*x6*cosx12-1.872*x5*x7*
        cosx2-1.633*x5*x10*cosx6-0.684*x6*x6*cosx7-4.561*
```

```
          x6*x11*cosx2
A(11,10)=3.226+0.921*x12-4.052*x4*x6+0.808*x4*x9-1.33*
          x7*x2+2.151*x7*x5+3.804*x7*x7+0.019*x7*x9+4.167*
          x12*x7-0.619*x7*cosx2+0.334*x7*cosx5-0.629*x7*
          cosx10
A(11,11)=4.41-2.705*x8-3.117*x3*x9+3.5*x2*x12*cosx6+
          3.246*x2*x12*cosx10-1.846*x8*x7*cosx10
A(12,2)=4.422-3.577*x7*cosx11+2.234*x8*cosx9
A(12,3)=3.517-0.841*x8*x4-3.077*x2*cosx4-4.508*x2*cosx6+
         1.462*x4*cosx2
A(12,5)=1.346+2.867*x9*x4+1.693*x2*cosx1-3.208*x2*cosx5+
         2.337*x2*cosx9-3.904*x12*cosx4+4.61*x12*cosx5+
         2.71*x12*cosx6+0.666*x12*cosx12+0.662*x7*x5*
         cosx5-1.368*x7*x9*cosx3
A(12,6)=1.791-2.802*x8-2.813*x9-0.125*x12-2.267*x4*x1+
         0.679*x8*x11+0.791*x11*x6+0.803*x11*x7-1.142*
         x11*x12-4.194*x6*cosx10+0.135*x7*cosx2-0.144*
         x7*cosx11+4.272*x2*x2*cosx5+4.624*x5*x11*cosx2
A(12,7)=4.568+1.297*x1+0.671*x3*x1-4.577*x12*x1*cosx10+
         2.514*x12*x1*cosx11+2.697*x12*x8*cosx10
A(12,8)=4.669+0.968*x8+0.205*x12+0.257*x4*cosx11+2.57*
         x7*cosx1-3.353*x7*cosx5-2.467*x7*cosx8+1.17*x7*
         cosx12-0.151*x10*cosx2-3.329*x10*cosx8+
A(12,9)=2.24+3.48*x7*x5+0.323*x3*x2*cosx1+0.101*x3*x2*
         cosx3-3.04*x3*x2*cosx12-1.584*x6*x2*cosx1+0.777*
         x6*x5*cosx5
A(12,11)=4.035-4.478*x6*x9-3.478*x6*x11-4.504*x8*cosx8+
          0.048*x3*x8*cosx7+1.374*x3*x8*cosx8-0.896*x3*
          x12*cosx3+3.52*x3*x12*cosx9+3.469*x10*x2*cosx4
b(1)=-0.023*x1+1.547*x2-0.066*x3+36.229*x4+0.015*x5+
      0.007* x6-166.955*x7+188.965*x8+0.009*x9+0.007*x10+
      0.009*x11+0.007*x12+2.116*u5+2.116*u6
b(2)=0.001*x1+0.002*x2+0.002*x3+0.004*x4-18.928*x5-0.009*
      x6-161.826*x7+123.607*x8+0.005*x9+0.007*x10+0.005*
      x11+0.007*x12+1.437*u5+1.738*u6
b(3)=0.006*x1+0.004*x2-23.187*x3+37.599*x4+0.009*x5+
      0.01*x6-0.005*x7+25.772*x8-375.205*x9+287.278*x10-
      1171.315*x11-0.089*x12+1.97*u5+1.97*u6
b(4)=-0.104*x1+5.762*x2-0.045*x3+33.578*x4-0.058*x5+
      102.109*x6-0.048*x7+147.311*x8-0.105*x9+593.476*
      x10-0.041*x11+714.566*x12+5.529*u5+3.904*u6
b(5)=0.005*x1+0.006*x2-16.482*x3-0.026*x4-0.042*x5+
      69.82*x6-246.683*x7+246.509*x8-391.95*x9-0.066*
      x10-364.839*x11+1172.63*x12+5.248*u5+4.137*u6
b(6)=-3.645*x1+1.774*x2+0.014*x3+0.015*x4-120.115*x5+
      148.685*x6-25.664*x7+27.031*x8-292.15*x9+151.982*
      x10-19.813*x11+0.006*x12+7.65*u5+7.192*u6
b(7)=-0.047*x1+2.56*x2-68.094*x3-0.135*x4-0.078*x5+
      102.087*x6+0.009*x7+0.005*x8-0.015*x9+93.825*x10+
      0.004*x11+0.003*x12+4.611*u5+2.986*u6
b(8)=-0.013*x1+0.842*x2-60.589*x3-0.11*x4-129.602*x5+
      20.513*x6-0.11*x7+238.06*x8+0.009*x9+21.64*x10-
      1389.987*x11+1026.681*x12+5.482*u5+7.217*u6
b(9)=-5.098*x1+2.431*x2-0.027*x3+17.977*x4+7.392*x5+
      0.022*x6-189.176*x7+228.412*x8-541.227*x9-0.1*
      x10+0.01*x11+0.012*x12+2.864*u5+2.747*u6
b(10)=-6.061*x1-0.116*x2-0.102*x3+55.843*x4-133.242*x5+
       29.038*x6-0.102*x7+238.524*x8-0.062*x9+384.886*x10-
       973.501*x11+657.053*x12+6.195*u5+7.851*u6
b(11)=-3.321*x1-0.063*x2+0.006*x3+0.013*x4-73.864*x5-
       0.051*x6-58.53*x7+190.545*x8-403.952*x9-0.075*x10-
       0.103*x11+1392.338*x12+3.196*u5+4.372*u6
b(12)=-5.482*x1-0.105*x2-0.106*x3+58.498*x4-66.77*x5+
       38.131*x6-233.454*x7+240.282*x8-0.039*x9+281.648*
       x10-0.086*x11+1279.236*x12+6.162*u5+6.617*u6
```

benchmark #2

This ODE system is characterized by:

1. 50% sparsity for the matrix A distributed linearly over its rows from 42% to 58%.

2. 90% sparsity for the matrices $\alpha, \beta, \gamma, \delta$ and θ of each nonzero element of A.

3. Linear terms ten times larger than the nonlinear terms in the nonzero elements of A.

4. 50 Hz highest frequency.

```
A(1,1)=2.526+4.085*x8*cosx10
A(1,3)=5.679-1.831*x6-1.211*x1*x9+2.082*x1*x11-2.248*x9*
       x5
A(1,7)=6.842-4.39*x2-3.39*x4*x4+0.121*x4*x10-1.124*x10*
       x2*cosx1+4.634*x10*x2*cosx7+0.159*x10*x10*cosx2+
       0.355*x10*x10*cosx4-2.437*x10*x10*cosx10+4.2*x10*
       x12*cosx6+0.951*x10*x12*cosx9
A(1,8)=6.637+0.003*x3*x9-3.12*x8*x5+1.617*x8*x8+3.048*x3*
       x9*cosx1-4.993*x3*x9*cosx2+1.141*x4*x11*cosx7+
       2.838*x7*x9*cosx6+2.779*x7*x12*cosx8
A(2,6)=1.239-0.415*x3-2.06*x9-1.275*x3*x1+1.674*x3*x4-
       0.147*x6*x5-1.132*x6*x8
A(2,7)=6.257+5.352*x3+5.281*x4+1.192*x1*x10-4.122*x1*
       cosx8
A(2,8)=6.383-9.543*x8-1.867*x1*x1-2.621*x6*x4+2.013*x6*
       x11+1.313*x5*cosx2-0.086*x5*cosx7+2.089*x8*cosx6-
       3.585*x8*x6*cosx3+3.133*x8*x6*cosx9
A(3,3)=3.164+5.457*x1-2.173*x2*cosx11-4.044*x2*cosx12-
       0.48*x6*x3*cosx3-4.934*x6*x10*cosx6
A(3,4)=3.627-9.697*x3-3.055*x11+1.799*x3*x2-1.218*x2*
       cosx11
A(3,7)=1.686+8.646*x2+0.948*x6+2.74*x1*x7+4.852*x3*x6-
       2.375*x2*cosx11
A(3,9)=4.574+4.289*x11*x2*cosx10-2.089*x11*x2*cosx12
A(3,10)=4.4-8.877*x2+6.014*x10+2.222*x1*x1+3.044*x1*
        x7-2.932*x3*cosx10+4.033*x3*cosx12
A(3,12)=5.319-6.984*x3+8.482*x10-2.138*x5*x4-2.174*x10*
        x9+2.785*x2*cosx4-4.509*x2*cosx7-4.579*x2*cosx10-
        4.193*x6*cosx10
A(4,1)=9.156-0.954*x3*x2*cosx11+3.576*x3*x5*cosx8-2.38*
       x3*x11*cosx7-4.374*x3*x11*cosx9-3.627*x3*x11*
       cosx12-4.411*x3*x12*cosx1-2.132*x3*x12*cosx5
A(4,3)=5.295+6.135*x7
A(4,5)=6.608+2.262*x6+2.5*x11*x1*cosx3-1.935*x11*x5*
       cosx2-1.621*x11*x5*cosx5-0.4*x11*x5*cosx7-1.026*
       x11*x5*cosx12
A(4,7)=5.856+4.542*x9*cosx5-2.742*x9*cosx7
A(4,9)=9.447-0.21*x5-6.593*x10+0.547*x5*x8*cosx6+2.642*
```

```
         x5*x8*cosx8-2.451*x8*x11*cosx8
A(4,11)=4.522-2.644*x1*x4+2.765*x9*x6
A(5,4)=2.75-4.498*x8-7.076*x9+0.949*x2*cosx6+4.454*x1*
         x1*cosx9-3.794*x1*x1*cosx11+3.172*x1*x10*cosx7+
         1.694*x2*x6*cosx1+4.72*x2*x6*cosx3+4.334*x6*x4*
         cosx10+1.782*x6*x4*cosx12
A(5,5)=4.382+6.182*x10+2.008*x10*x2+0.292*x6*cosx9-4.489*
         x1*x6*cosx11-1.898*x1*x6*cosx12
A(5,7)=9.809-1.104*x11-1.422*x5*x4+3.355*x5*x11+1.073*x9*
         x12*cosx9
A(5,8)=9.791-7.412*x11-2.876*x1*x4*cosx2+0.457*x1*x4*
         cosx10+1.999*x1*x12*cosx11
A(5,10)=6.273+7.242*x10-1.608*x7*x6-4.298*x7*x7+0.106*
          x4*cosx1+1.216*x9*cosx4-1.484*x9*cosx10+3.01*
          x5*x4*cosx3+2.382*x5*x4*cosx7
A(5,11)=5.949-7.896*x2-6.324*x6-2.878*x11+4.533*x3*x4-
          1.551*x3*x5+3.723*x3*x8
A(5,12)=2.968+8.42*x6+4.542*x1*x2-3.625*x1*x7+2.92*
          x12*cosx1+0.979*x12*cosx8
A(6,1)=4.76+6.193*x2-5.803*x11-2.654*x1*cosx3-0.067*x11*
         cosx11-4.153*x12*cosx12
A(6,2)=2.632+8.213*x1-8.525*x2-8.02*x6-9.59*x8
A(6,5)=9.687-4.396*x5+3.613*x4*x11+0.059*x5*x1+1.062*x5*
         x8-2.354*x9*cosx8-3.839*x10*cosx5+3.284*x10*cosx10
A(6,6)=6.763-5.997*x3+2.207*x1*cosx2+4.064*x1*cosx7-
         3.002*x1*cosx8+0.933*x9*cosx8+1.388*x9*cosx10
A(6,7)=0.665-4.8*x6*cosx9+0.502*x6*cosx11
A(6,8)=1.002+4.523*x12*x11+0.599*x5*x5*cosx9+3.597*x5*x5*
         cosx11-3.203*x5*x9*cosx7+3.016*x11*x5*cosx12-
         1.839*x11*x12*cosx2-2.023*x11*x12*cosx3+0.428*x11*
         x12*cosx7
A(6,9)=2.412-2.839*x4*x3-4.841*x7*x4*cosx10+1.587*x7*x4*
         cosx12
A(6,10)=4.817-3.705*x2+2.628*x3-4.659*x1*x8+2.769*x2*
          cosx3+2.24*x1*x3*cosx11
A(6,12)=0.374+5.09*x6-4.029*x10+2.325*x4*x3*cosx9+3.3*
          x4*x3*cosx12
A(7,1)=4.056+3.847*x11*x4-3.68*x11*x10-0.788*x5*x4*cosx1-
         3.327*x5*x12*cosx1+1.794*x5*x12*cosx8
A(7,4)=8.435-0.794*x1*cosx1+4.689*x1*cosx10-1.105*x2*
         cosx3-2.536*x2*cosx7+0.054*x11*cosx6+0.937*x3*
         x5*cosx2+2.916*x3*x7*cosx1+1.275*x3*x7*cosx11-
         3.145*x7*x3*cosx3-1.256*x8*x12*cosx9+1.624*x10*
         x7*cosx11
A(7,5)=6.57+2.114*x4-1.699*x1*x4-4.7*x9*x8-0.421*x1*
         x1*cosx4
A(7,9)=1.492+1.532*x4-1.061*x6+2.335*x9*x2+1.129*x8*x6*
         cosx7-1.026*x8*x6*cosx10
A(8,1)=2.727+9.028*x2
A(8,4)=9.713-6.469*x7-1.157*x4*x2+4.886*x12*cosx6-1.143*
         x12*cosx12
A(8,5)=1.316+0.013*x3*x2-2.112*x3*x5-4.419*x3*x8
A(8,6)=8.68-3.551*x9+3.247*x2*cosx1+2.354*x5*cosx4+
         4.087*x5*cosx5+1.625*x1*x6*cosx2
A(8,7)=9.53-0.669*x5*x5+1.316*x5*x8-2.196*x7*cosx3-
         1.237*x7*cosx6
A(8,9)=0.805-4.816*x5*cosx2+1.338*x5*cosx4-1.341*x5*cosx5
A(8,11)=6.534+3.326*x11-2.614*x11*x8+2.996*x11*x11-0.77*
          x7*x4*cosx9-4.095*x7*x12*cosx4+0.86*x7*x12*
          cosx12+1.083*x11*x9*cosx4
A(8,12)=8.85-0.316*x6*x6*cosx11
A(9,1)=4.059+2.545*x7*cosx2-4.479*x7*cosx10-0.242*x9*
         cosx8+0.432*x9*cosx11
A(9,2)=8.069+9.001*x10+2.915*x3*x3+2.749*x3*x6-0.336*x3*
         x10-0.432*x5*x1
A(9,3)=3.514-3.398*x5+3.31*x1*x11+2.536*x5*x6+2.815*x5*
         x11+4.39*x6*x7+4.155*x4*cosx1+2.393*x4*cosx6+
         1.036*x4*cosx8-4.526*x10*cosx5+3.44*x10*cosx11-
```

```
          0.145*x5*x6*cosx2
A(9,6)=0.16+8.05*x2-7.216*x6-0.469*x10*x2+3.442*x12*
          x9+0.885*x1*cosx3-1.239*x9*cosx5
A(9,7)=9.169-2.046*x3+0.946*x5*x3*cosx3-2.261*x5*x3*
          cosx8+0.719*x5*x5*cosx6-2.713*x5*x5*cosx11-
          0.451*x5*x8*cosx6-0.209*x5*x8*cosx9
A(9,8)=7.662-8.858*x3+1.901*x2*x2-3.242*x2*x8+3.685*x7*
          cosx1
A(9,10)=8.614
A(10,2)=9.74+7.878*x4-2.208*x6+8.883*x11+1.052*x2*x3+
           4.111*x2*x4-2.406*x2*x5+0.686*x5*x6*cosx1-0.423*
           x5*x6*cosx7-4.826*x5*x8*cosx1-3.67*x5*x8*cosx2
A(10,3)=8.38+3.502*x1-8.314*x9-3.008*x1*x6-1.028*x5*
           cosx5-0.985*x12*x1*cosx4+4.156*x12*x2*cosx8+
           3.692*x12*x2*cosx11-1.439*x12*x2*cosx12-2.557*
           x12*x3*cosx12-4.437*x12*x12*cosx11
A(10,5)=1.085-4.956*x2+1.096*x10-2.257*x11*cosx7-2.629*
           x3*x8*cosx12
A(10,6)=7.245+4.987*x2+1.759*x9-3.779*x2*x4+0.39*x2*x10+
           1.97*x12*x11-3.886*x2*cosx7-2.411*x2*cosx11-
           0.273*x2*cosx12+4.315*x10*x10*cosx5-1.897*x12*x4*
           cosx3+1.792*x12*x4*cosx7-4.947*x12*x10*cosx3-
           3.579*x12*x10*cosx9+4.866*x12*x10*cosx10
A(10,7)=9.49
A(10,9)=6.453-1.116*x8-8.929*x11+3.403*x6*cosx3+2.376*
           x10*cosx7+2.66*x4*x8*cosx12
A(10,11)=4.77+8.293*x5-7.791*x12-0.134*x1*cosx5-2.872*
            x1*cosx10-0.677*x1*cosx12+2.169*x5*cosx6
A(10,12)=6.169-4.043*x8+2.2*x1*x8*cosx7-0.552*x1*x8*
            cosx11
A(11,2)=5.247-1.345*x2-3.635*x4+4.06*x4*cosx8-2.807*x9*
           cosx7-4.203*x9*cosx10-0.142*x3*x10*cosx12
A(11,6)=4.456+3.211*x1+4.086*x2-0.336*x7+9.143*x10+9.784*
           x12-3.988*x6*cosx2+2.243*x9*cosx10+2.193*x12*x5*
           cosx4+3.616*x12*x5*cosx6
A(11,7)=6.709+5.482*x1-1.648*x4+4.642*x3*x6+4.292*x5*x2+
           0.843*x5*x4+3.607*x9*cosx9-0.964*x3*x8*cosx10
A(11,8)=2.698-4.058*x3*cosx8+1.153*x6*cosx12-1.872*x5*x7*
           cosx2-1.633*x5*x10*cosx6-0.684*x6*x6*cosx7-4.561*
           x6*x11*cosx2
A(11,10)=6.452+1.842*x12-4.052*x4*x6+0.808*x4*x9-1.33*
            x7*x2+2.151*x7*x5+3.804*x7*x7+0.019*x7*x9+4.167*
            x12*x7-0.619*x7*cosx2+0.334*x7*cosx5-0.629*x7*
            cosx10
A(11,11)=8.82-5.411*x8-3.117*x3*x9+3.5*x2*x12*cosx6+
            3.246*x2*x12*cosx10-1.846*x8*x7*cosx10
A(12,2)=8.843-3.577*x7*cosx11+2.234*x8*cosx9
A(12,3)=7.033-0.841*x8*x4-3.077*x2*cosx4-4.508*x2*cosx6+
           1.462*x4*cosx2
A(12,5)=2.692+2.867*x9*x4+1.693*x2*cosx1-3.208*x2*cosx5+
           2.337*x2*cosx9-3.904*x12*cosx4+4.61*x12*cosx5+
           2.71*x12*cosx6+0.666*x12*cosx12+0.662*x7*x5*
           cosx5-1.368*x7*x9*cosx3
A(12,6)=3.583-5.604*x8-5.625*x9-0.249*x12-2.267*x4*x1+
           0.679*x8*x11+0.791*x11*x6+0.803*x11*x7-1.142*x11*
           x12-4.194*x6*cosx10+0.135*x7*cosx2-0.144*x7*
           cosx11+4.272*x2*x2*cosx5+4.624*x5*x11*cosx2
A(12,7)=9.137+2.594*x1+0.671*x3*x1-4.577*x12*x1*cosx10+
           2.514*x12*x1*cosx11+2.697*x12*x8*cosx10
A(12,8)=9.339+1.936*x8+0.41*x12+0.257*x4*cosx11+2.57*
           x7*cosx1-3.353*x7*cosx5-2.467*x7*cosx8+1.17*x7*
           cosx12-0.151*x10*cosx2-3.329*x10*cosx8
A(12,9)=4.481+3.48*x7*x5+0.323*x3*x2*cosx1+0.101*x3*x2*
           cosx3-3.04*x3*x2*cosx12-1.584*x6*x2*cosx1+0.777*
           x6*x5*cosx5
A(12,11)=8.07-4.478*x6*x9-3.478*x6*x11-4.504*x8*cosx8+
            0.048*x3*x8*cosx7+1.374*x3*x8*cosx8-0.896*x3*
            x12*cosx3+3.52*x3*x12*cosx9+3.469*x10*x2*cosx4
```

```
b(1)=-0.047*x1+3.135*x2-0.132*x3+72.46*x4+0.029*x5+
      0.013*x6-333.85*x7+378.058*x8+0.018*x9+0.013*x10+
      0.018*x11+0.013*x12+4.235*u5+4.235*u6
b(2)=0.003*x1+0.004*x2+0.006*x3+0.009*x4-38.119*x5-0.017*
      x6-324.237*x7+280.848*x8+0.011*x9+0.014*x10+0.011*
      x11+0.014*x12+
      3.015*u5+3.621*u6
b(3)=0.011*x1+0.008*x2-44.073*x3+63.003*x4+0.016*x5+
      0.019*x6-0.002*x7+33.099*x8-738.155*x9+574.585*x10-
      1969.674*x11-0.149*x12+3.507*u5+3.507*u6
b(4)=-0.208*x1+11.512*x2-0.089*x3+67.157*x4-0.117*x5+
      204.214*x6-0.096*x7+294.478*x8-0.21*x9+1187.103*
      x10-0.082*x11+1424.913*x12+11.056*u5+7.806*u6
b(5)=0.01*x1+0.011*x2-33.902*x3-0.053*x4-0.083*x5+
      138.51*x6-492.75*x7+492.905*x8-785.363*x9-0.133*
      x10-768.064*x11+2347.739*x12+10.486*u5+8.283*u6
b(6)=-7.301*x1+4.086*x2+0.028*x3+0.029*x4-223.454*x5+
      297.475*x6-50.850*x7+43.753*x8-636.412*x9+303.489*
      x10-41.416*x11+0.011*x12+15.076*u5+13.892*u6
b(7)=-0.095*x1+5.115*x2-121.082*x3-0.241*x4-0.157*x5+
      204.243*x6+0.018*x7+0.01*x8-0.03*x9+187.717*x10+
      0.008*x11+0.005*x12+8.983*u5+5.733*u6
b(8)=-0.026*x1+1.684*x2-121.28*x3-0.22*x4-265.854*x5+
      41.088*x6-0.22*x7+477.596*x8+0.013*x9+72.21*x10-
      2780.2*x11+2053.291*x12+11.103*u5+14.678*u6
b(9)=-10.201*x1+4.881*x2-0.06*x3+39.047*x4+19.227*x5+
      0.047*x6-374.751*x7+457.142*x8-1082.454*x9-0.2*
      x10+0.02*x11+0.024*x12+5.696*u5+5.391*u6
b(10)=-12.078*x1-0.233*x2-0.204*x3+111.998*x4-234.857*x5+
       58.125*x6-0.203*x7+477.046*x8-0.127*x9+790.567*x10-
       1947.684*x11+1469.275*x12+11.954*u5+14.764*u6
b(11)=-6.819*x1-0.129*x2+0.012*x3+0.026*x4-141.644*x5-
       0.096*x6-126.426*x7+382.813*x8-809.258*x9-0.151*
       x10-0.206*x11+2784.478*x12+6.354*u5+8.609*u6
b(12)=-11.024*x1-0.211*x2-0.183*x3+102.704*x4-123.619*
       x5+ 80.381*x6-467.873*x7+480.377*x8-0.078*x9+563.2*
       x10-0.172*x11+2546.93*x12+12.074*u5+12.761*u6
```

benchmark #3

This ODE system is characterized by:

1. 50% sparsity for the matrix A distributed linearly over its rows from 42% to 58%.
2. 90% sparsity for the matrices $\alpha, \beta, \gamma, \delta$ and θ of each nonzero element of A.
3. Linear terms ten times larger than the nonlinear terms in the nonzero elements of A.
4. 90 Hz highest frequency.

```
A(1,1)=2.526+4.085*x8*cosx10
A(1,3)=5.679-1.831*x6-1.211*x1*x9+2.082*x1*x11-2.248*x9*
        x5
A(1,7)=6.842-4.39*x2-3.39*x4*x4+0.121*x4*x10-1.124*x10*
        x2*cosx1+4.634*x10*x2*cosx7+0.159*x10*x10*cosx2+
        0.355*x10*x10*cosx4-2.437*x10*x10*cosx10+4.2*x10*
        x12*cosx6+0.951*x10*x12*cosx9
A(1,8)=6.637+0.003*x3*x9-3.12*x8*x5+1.617*x8*x8+3.048*x3*
        x9*cosx1-4.993*x3*x9*cosx2+1.141*x4*x11*cosx7+
        2.838*x7*x9*cosx6+2.779*x7*x12*cosx8
A(2,6)=1.239-0.415*x3-2.06*x9-1.275*x3*x1+1.674*x3*x4-
        0.147*x6*x5-1.132*x6*x8
A(2,7)=6.257+5.352*x3+5.281*x4+1.192*x1*x10-4.122*x1*
        cosx8
A(2,8)=6.383-9.543*x8-1.867*x1*x1-2.621*x6*x4+2.013*x6*
        x11+1.313*x5*cosx2-0.086*x5*cosx7+2.089*x8*cosx6-
        3.585*x8*x6*cosx3+3.133*x8*x6*cosx9
A(3,3)=3.164+5.457*x1-2.173*x2*cosx11-4.044*x2*cosx12-
        0.48*x6*x3*cosx3-4.934*x6*x10*cosx6
A(3,4)=3.627-9.697*x3-3.055*x11+1.799*x3*x2-1.218*x2*
        cosx11
A(3,7)=1.686+8.646*x2+0.948*x6+2.74*x1*x7+4.852*x3*x6-
        2.375*x2*cosx11
A(3,9)=4.574+4.289*x11*x2*cosx10-2.089*x11*x2*cosx12
A(3,10)=4.4-8.877*x2+6.014*x10+2.222*x1*x1+3.044*x1*
         x7-2.932*x3*cosx10+4.033*x3*cosx12
A(3,12)=5.319-6.984*x3+8.482*x10-2.138*x5*x4-2.174*x10*
         x9+2.785*x2*cosx4-4.509*x2*cosx7-4.579*x2*cosx10-
         4.193*x6*cosx10
A(4,1)=9.156-0.954*x3*x2*cosx11+3.576*x3*x5*cosx8-2.38*
        x3*x11*cosx7-4.374*x3*x11*cosx9-3.627*x3*x11*
        cosx12-4.411*x3*x12*cosx1-2.132*x3*x12*cosx5
A(4,3)=5.295+6.135*x7
A(4,5)=6.608+2.262*x6+2.5*x11*x1*cosx3-1.935*x11*x5*
        cosx2-1.621*x11*x5*cosx5-0.4*x11*x5*cosx7-1.026*
        x11*x5*cosx12
A(4,7)=5.856+4.542*x9*cosx5-2.742*x9*cosx7
A(4,9)=9.447-0.21*x5-6.593*x10+0.547*x5*x8*cosx6+2.642*
        x5*x8*cosx8-2.451*x8*x11*cosx8
A(4,11)=4.522-2.644*x1*x4+2.765*x9*x6
A(5,4)=2.75-4.498*x8-7.076*x9+0.949*x2*cosx6+4.454*x1*
        x1*cosx9-3.794*x1*x1*cosx11+3.172*x1*x10*cosx7+
        1.694*x2*x6*cosx1+4.72*x2*x6*cosx3+4.334*x6*x4*
        cosx10+1.782*x6*x4*cosx12
A(5,5)=4.382+6.182*x10+2.008*x10*x2+0.292*x6*cosx9-4.489*
        x1*x6*cosx11-1.898*x1*x6*cosx12
A(5,7)=9.809-1.104*x11-1.422*x5*x4+3.355*x5*x11+1.073*x9*
        x12*cosx9
A(5,8)=9.791-7.412*x11-2.876*x1*x4*cosx2+0.457*x1*x4*
        cosx10+1.999*x1*x12*cosx11
A(5,10)=6.273+7.242*x10-1.608*x7*x6-4.298*x7*x7+0.106*
         x4*cosx1+1.216*x9*cosx4-1.484*x9*cosx10+3.01*
         x5*x4*cosx3+2.382*x5*x4*cosx7
A(5,11)=5.949-7.896*x2-6.324*x6-2.878*x11+4.533*x3*x4-
         1.551*x3*x5+3.723*x3*x8
A(5,12)=2.968+8.42*x6+4.542*x1*x2-3.625*x1*x7+2.92*
         x12*cosx1+0.979*x12*cosx8
A(6,1)=4.76+6.193*x2-5.803*x11-2.654*x1*cosx3-0.067*x11*
        cosx11-4.153*x12*cosx12
A(6,2)=2.632+8.213*x1-8.525*x2-8.02*x6-9.59*x8
A(6,5)=9.687-4.396*x5+3.613*x4*x11+0.059*x5*x1+1.062*x5*
        x8-2.354*x9*cosx8-3.839*x10*cosx5+3.284*x10*cosx10
A(6,6)=6.763-5.997*x3+2.207*x1*cosx2+4.064*x1*cosx7-
        3.002*x1*cosx8+0.933*x9*cosx8+1.388*x9*cosx10
A(6,7)=0.665-4.8*x6*cosx9+0.502*x6*cosx11
A(6,8)=1.002+4.523*x12*x11+0.599*x5*x5*cosx9+3.597*x5*x5*
        cosx11-3.203*x5*x9*cosx7+3.016*x11*x5*cosx12-
        1.839*x11*x12*cosx2-2.023*x11*x12*cosx3+0.428*x11*
        x12*cosx7
A(6,9)=2.412-2.839*x4*x3-4.841*x7*x4*cosx10+1.587*x7*x4*
        cosx12
A(6,10)=4.817-3.705*x2+2.628*x3-4.659*x1*x8+2.769*x2*
         cosx3+2.24*x1*x3*cosx11
A(6,12)=0.374+5.09*x6-4.029*x10+2.325*x4*x3*cosx9+3.3*
         x4*x3*cosx12
A(7,1)=4.056+3.847*x11*x4-3.68*x11*x10-0.788*x5*x4*cosx1-
        3.327*x5*x12*cosx1+1.794*x5*x12*cosx8
A(7,4)=8.435-0.794*x1*cosx1+4.689*x1*cosx10-1.105*x2*
        cosx3-2.536*x2*cosx7+0.054*x11*cosx6+0.937*x3*
        x5*cosx2+2.916*x3*x7*cosx1+1.275*x3*x7*cosx11-
        3.145*x7*x3*cosx3-1.256*x8*x12*cosx9+1.624*x10*
        x7*cosx11
A(7,5)=6.57+2.114*x4-1.699*x1*x4-4.7*x9*x8-0.421*x1*
        x1*cosx4
A(7,9)=1.492+1.532*x4-1.061*x6+2.335*x9*x2+1.129*x8*x6*
        cosx7-1.026*x8*x6*cosx10
A(8,1)=2.727+9.028*x2
A(8,4)=9.713-6.469*x7-1.157*x4*x2+4.886*x12*cosx6-1.143*
        x12*cosx12
A(8,5)=1.316+0.013*x3*x2-2.112*x3*x5-4.419*x3*x8
A(8,6)=8.68-3.551*x9+3.247*x2*cosx1+2.354*x5*cosx4+
        4.087*x5*cosx5+1.625*x1*x6*cosx2
A(8,7)=9.53-0.669*x5*x5+1.316*x5*x8-2.196*x7*cosx3-
        1.237*x7*cosx6
A(8,9)=0.805-4.816*x5*cosx2+1.338*x5*cosx4-1.341*x5*cosx5
A(8,11)=6.534+3.326*x11-2.614*x11*x8+2.996*x11*x11-0.77*
         x7*x4*cosx9-4.095*x7*x12*cosx4+0.86*x7*x12*
         cosx12+1.083*x11*x9*cosx4
A(8,12)=8.85-0.316*x6*x6*cosx11
A(9,1)=4.059+2.545*x7*cosx2-4.479*x7*cosx10-0.242*x9*
        cosx8+0.432*x9*cosx11
A(9,2)=8.069+9.001*x10+2.915*x3*x3+2.749*x3*x6-0.336*x3*
        x10-0.432*x5*x1
A(9,3)=3.514-3.398*x5+3.31*x1*x11+2.536*x5*x6+2.815*x5*
        x11+4.39*x6*x7+4.155*x4*cosx1+2.393*x4*cosx6+
        1.036*x4*cosx8-4.526*x10*cosx5+3.44*x10*cosx11-
        0.145*x5*x6*cosx2
A(9,6)=0.16+8.05*x2-7.216*x6-0.469*x10*x2+3.442*x12*
        x9+0.885*x1*cosx3-1.239*x9*cosx5
A(9,7)=9.169-2.046*x3+0.946*x5*x3*cosx3-2.261*x5*x3*
        cosx8+0.719*x5*x5*cosx6-2.713*x5*x5*cosx11-
        0.451*x5*x8*cosx6-0.209*x5*x8*cosx9
A(9,8)=7.662-8.858*x3+1.901*x2*x2-3.242*x2*x8+3.685*x7*
        cosx1
A(9,10)=8.614
A(10,2)=9.74+7.878*x4-2.208*x6+8.883*x11+1.052*x2*x3+
         4.111*x2*x4-2.406*x2*x5+0.686*x5*x6*cosx1-0.423*
         x5*x6*cosx7-4.826*x5*x8*cosx1-3.67*x5*x8*cosx2
A(10,3)=8.38+3.502*x1-8.314*x9-3.008*x1*x6-1.028*x5*
         cosx5-0.985*x12*x1*cosx4+4.156*x12*x2*cosx8+
         3.692*x12*x2*cosx11-1.439*x12*x2*cosx12-2.557*
         x12*x3*cosx12-4.437*x12*x12*cosx11
A(10,5)=1.085-4.956*x2+1.096*x10-2.257*x11*cosx7-2.629*
         x3*x8*cosx12
A(10,6)=7.245+4.987*x2+1.759*x9-3.779*x2*x4+0.39*x2*x10+
         1.97*x12*x11-3.886*x2*cosx7-2.411*x2*cosx11-
         0.273*x2*cosx12+4.315*x10*x10*cosx5-1.897*x12*x4*
         cosx3+1.792*x12*x4*cosx7-4.947*x12*x10*cosx3-
         3.579*x12*x10*cosx9+4.866*x12*x10*cosx10
A(10,7)=9.49
A(10,9)=6.453-1.116*x8-8.929*x11+3.403*x6*cosx3+2.376*
         x10*cosx7+2.66*x4*x8*cosx12
A(10,11)=4.77+8.293*x5-7.791*x12-0.134*x1*cosx5-2.872*
          x1*cosx10-0.677*x1*cosx12+2.169*x5*cosx6
A(10,12)=6.169-4.043*x8+2.2*x1*x8*cosx7-0.552*x1*x8*
```

```
          cosx11
A(11,2)=5.247-1.345*x2-3.635*x4+4.06*x4*cosx8-2.807*x9*
         cosx7-4.203*x9*cosx10-0.142*x3*x10*cosx12
A(11,6)=4.456+3.211*x1+4.086*x2-0.336*x7+9.143*x10+9.784*
         x12-3.988*x6*cosx2+2.243*x9*cosx10+2.193*x12*x5*
         cosx4+3.616*x12*x5*cosx6
A(11,7)=6.709+5.482*x1-1.648*x4+4.642*x3*x6+4.292*x5*x2+
         0.843*x5*x4+3.607*x9*cosx9-0.964*x3*x8*cosx10
A(11,8)=2.698-4.058*x3*cosx8+1.153*x6*cosx12-1.872*x5*x7*
         cosx2-1.633*x5*x10*cosx6-0.684*x6*x6*cosx7-4.561*
         x6*x11*cosx2
A(11,10)=6.452+1.842*x12-4.052*x4*x6+0.808*x4*x9-1.33*
          x7*x2+2.151*x7*x5+3.804*x7*x7+0.019*x7*x9+4.167*
          x12*x7-0.619*x7*cosx2+0.334*x7*cosx5-0.629*x7*
          cosx10
A(11,11)=8.82-5.411*x8-3.117*x3*x9+3.5*x2*x12*cosx6+
          3.246*x2*x12*cosx10-1.846*x8*x7*cosx10
A(12,2)=8.843-3.577*x7*cosx11+2.234*x8*cosx9
A(12,3)=7.033-0.841*x8*x4-3.077*x2*cosx4-4.508*x2*cosx6+
         1.462*x4*cosx2
A(12,5)=2.692+2.867*x9*x4+1.693*x2*cosx1-3.208*x2*cosx5+
         2.337*x2*cosx9-3.904*x12*cosx4+4.61*x12*cosx5+
         2.71*x12*cosx6+0.666*x12*cosx12+0.662*x7*x5*
         cosx5-1.368*x7*x9*cosx3
A(12,6)=3.583-5.604*x8-5.625*x9-0.249*x12-2.267*x4*x1+
         0.679*x8*x11+0.791*x11*x6+0.803*x11*x7-1.142*x11*
         x12-4.194*x6*cosx10+0.135*x7*cosx2-0.144*x7*
         cosx11+4.272*x2*x2*cosx5+4.624*x5*x11*cosx2
A(12,7)=9.137+2.594*x1+0.671*x3*x1-4.577*x12*x1*cosx10+
         2.514*x12*x1*cosx11+2.697*x12*x8*cosx10
A(12,8)=9.339+1.936*x8+0.41*x12+0.257*x4*cosx11+2.57*
         x7*cosx1-3.353*x7*cosx5-2.467*x7*cosx8+1.17*x7*
         cosx12-0.151*x10*cosx2-3.329*x10*cosx8
A(12,9)=4.481+3.48*x7*x5+0.323*x3*x2*cosx1+0.101*x3*x2*
         cosx3-3.04*x3*x2*cosx12-1.584*x6*x2*cosx1+0.777*
         x6*x5*cosx5
A(12,11)=8.07-4.478*x6*x9-3.478*x6*x11-4.504*x8*cosx8+
          0.048*x3*x8*cosx7+1.374*x3*x8*cosx8-0.896*x3*
          x12*cosx3+3.52*x3*x12*cosx9+3.469*x10*x2*cosx4
b(1)=-0.048*x1+3.176*x2-0.132*x3+72.098*x4+0.028*x5+
      0.013*x6-333.773*x7+343.75*x8+0.017*x9+0.013*x10+
      0.017*x11+0.013*x12
b(2)=0.003*x1+0.004*x2+0.006*x3+0.009*x4-38.904*x5-0.018*
      x6-322.971*x7+314.353*x8+0.013*x9+0.014*x10+0.013*
      x11+0.014*x12+1.239*u6
b(3)=0.008*x1+0.008*x2-45.656*x3+39.674*x4+0.014*x5+
      0.017*x6-0.029*x7+83.24*x8-553.007*x9+574.648*x10-
      3083.104*x11-0.13*x12
b(4)=-0.207*x1+11.499*x2-0.088*x3+66.543*x4-0.118*x5+
      205.341*x6-0.096*x7+294.334*x8-0.21*x9+1186.356*
      x10-0.082*x11+2557.248*x12+6.536*u5
b(5)=0.01*x1+0.011*x2-34.532*x3-0.054*x4-0.084*x5+
      137.395*x6-492.373*x7+492.827*x8-788.275*x9-0.133*
      x10-1527.211*x11+3477.64*x12+4.372*u5
b(6)=-3.735*x1+5.914*x2+0.031*x3+0.023*x4-212.68*x5+
      299.77*x6-50.587*x7+40.313*x8-605.325*x9+303.029*
      x10-119.958*x11+0.009*x12+9.547*u5+6.763*u6
b(7)=-0.095*x1+5.112*x2-105.977*x3-0.211*x4-0.159*x5+
      206.424*x6+0.018*x7+0.008*x8-0.031*x9+191.784*x10+
      0.009*x11+0.004*x12+6.57*u5
b(8)=-0.061*x1+3.454*x2-122.026*x3-0.221*x4-279.123*x5+
      41.145*x6-0.22*x7+479.038*x8+0.011*x9+81.851*x10-
      5004.469*x11+3694.767*x12+1.316*u5+8.885*u6
b(9)=-10.225*x1+4.901*x2-0.068*x3+42.792*x4-12.193*x5+
      0.022*x6-385.34*x7+460.631*x8-1082.454*x9-0.199*
      x10+0.02*x11+0.025*x12+0.389*u6
b(10)=-12.298*x1-0.237*x2-0.191*x3+104.927*x4-227.578*x5+
       33.913*x6-0.204*x7+477.046*x8-0.129*x9+797.359*x10-
       3488.785*x11+2885.762*x12+1.085*u5+7.245*u6
b(11)=-6.587*x1-0.125*x2+0.011*x3+0.026*x4-143.945*x5-
       0.098*x6-133.891*x7+337.185*x8-810.76*x9-0.151*
       x10-0.207*x11+4987.344*x12+4.583*u6
b(12)=-11.084*x1-0.213*x2-0.155*x3+88.414*x4-116.766*x5+
       84.497*x6-469.615*x7+459.031*x8-0.079*x9+563.005*
       x10-0.173*x11+4563.684*x12+2.692*u5+3.716*u6
```

Chapter 7 suggests that the following four benchmarks are required for the analysis of the multiprocessor implementation procedure:

benchmark #1

This ODE system is characterized by:

1. 95% sparsity for the matrix A distributed linearly over its rows from 92% to 98%.

2. 99% sparsity for the matrices $\alpha, \beta, \gamma, \delta$ and θ of each nonzero element of A.

3. Linear terms as large as the nonlinear terms in the nonzero elements of A.

4. 50 Hz highest frequency.

```
A(1,39)=1.235+1.532*x48
A(2,2)=3.152
A(4,39)=0.171+4.66*x17
A(5,20)=3.406+4.246*x36
A(5,36)=0.611
A(5,44)=3.708+0.621*x14*x10-2.499*x41*x5*cosx37
A(6,30)=1.75-4.032*x36
A(7,31)=0.556
A(8,15)=0.035
A(8,36)=2.909+0.236*x35
A(9,27)=3.451-2.108*x7-3.349*x34
A(9,31)=1.733
A(10,19)=4.43
A(10,33)=3.012-0.877*x19-0.477*x27+2.098*x36*x27+0.963*
         x14*x30*cosx3-1.355*x14*x30*cosx42
A(10,44)=2.141+2.627*x27*x30-0.265*x45*x8
A(11,4)=2.024-4.834*x26*cosx22
A(11,22)=4.759
A(11,36)=0.719+1.571*x9
A(12,16)=0.845
A(12,43)=1.039-0.824*x9*x2
A(13,20)=3.722+2.87*x40*x39*cosx27-0.921*x40*x39*cosx48
A(13,35)=0.743
A(14,13)=3.017
A(16,43)=1.891
```

```
A(17,5)=0.931
A(17,14)=4.356-0.723*x32+4.646*x12*x29
A(17,21)=2.812-3.276*x26+4.908*x44
A(18,24)=1.693+3.233*x19+1.906*x15*x7-0.585*x15*x16
A(18,39)=3.254
A(19,17)=0.992-0.346*x32+4.125*x26*x23-1.179*x26*x32
A(19,34)=1.736+3.853*x7
A(19,40)=4.617-3.887*x26+2.108*x25*cosx37
A(20,1)=3.648
A(20,15)=2.863-3.033*x42*x14
A(20,42)=2.698-0.467*x39-2.941*x16*x27
A(20,43)=3.183
A(21,4)=0.484
A(21,38)=3.971+4.862*x2-2.77*x7+2.352*x33
A(22,5)=3.109
A(22,13)=3.205+2.063*x29
A(23,5)=0.072+3.105*x7*cosx42
A(23,38)=1.976
A(23,42)=3.95-0.75*x37
A(25,36)=3.745-0.939*x21+2.378*x27-2.687*x12*x9-0.895*
        x12*x34
A(26,4)=4.984-2.4*x3-1.904*x17
A(26,10)=1.436
A(26,36)=3.181+2.841*x7+4.938*x29*x25
A(27,6)=0.138-0.61*x13-0.132*x28+3.768*x30+1.429*x48
A(28,1)=1.102-1.794*x25+2.724*x26+2.406*x46*cosx8
A(28,8)=3.277+1.005*x30*x43
A(28,11)=4.408
A(28,18)=2.259
A(28,35)=2.307+0.546*x47*x48
A(28,38)=4.365
A(28,48)=2.909-1.828*x30+3.637*x22*x38
A(29,4)=4.814+3.86*x37
A(29,17)=0.299
A(29,34)=3.126+3.758*x47
A(29,46)=1.938
A(29,47)=4.3-3.039*x7-4.69*x19*cosx13
A(30,2)=4.52+0.33*x2+3.824*x17-3.881*x48*cosx9
A(30,47)=1.696
A(31,2)=2.588
A(31,6)=4.76+1.758*x2*cosx33
A(31,7)=2.238+1.229*x48*x27
A(31,28)=3.467
A(31,36)=1.832-0.078*x20+3.199*x19*cosx22
A(32,2)=0.432
A(32,3)=1.447
A(32,4)=1.706-1.245*x13
A(33,40)=0.223-2.438*x12+3.477*x26*cosx29-1.035*x26*
        cosx37
A(34,3)=1.673+0.353*x28*x29*cosx26
A(34,15)=3.567+2.849*x7
A(34,47)=3.329
A(35,28)=2.33-3.529*x7*cosx38
A(35,29)=2.101+1.878*x37*x44+4.075*x40*x39+0.771*x47*
        cosx42
A(35,48)=1.271
A(36,1)=3.695-3.595*x25-2.522*x37
A(36,5)=1.673-0.954*x2*x9*cosx8
A(36,20)=0.811-3.323*x19
A(36,26)=3.101
A(36,36)=4.217-2.037*x38*x14
A(36,44)=3.321+4.356*x14
A(37,12)=4.472+1.389*x35*x11
A(37,42)=2.047
A(37,45)=4.676-4.546*x12
A(38,18)=0.255
A(38,32)=0.531
A(39,7)=2.43
```

```
A(39,16)=1.651
A(39,32)=0.49
A(40,6)=2.682
A(40,13)=2.829
A(40,23)=4.063-2.729*x8
A(40,29)=4.753
A(41,35)=1.475-1.682*x11
A(41,38)=1.928-1.049*x8*x3
A(42,29)=4.283
A(42,42)=1.945
A(42,46)=2.267-0.412*x29-2.633*x12*cosx10
A(43,31)=2.194+3.991*x26
A(43,33)=3.207-3.827*x18
A(44,13)=3.749-4.864*x40*cosx30
A(44,22)=3.552-2.275*x26-3.151*x7*cosx18-4.166*x7*cosx26+
         4.714*x7*cosx39
A(44,30)=4.572+2.147*x8-4.986*x7*x33-0.663*x10*x38+0.525*
         x4*cosx17
A(44,38)=0.001-0.109*x3*x19
A(45,2)=4.611
A(45,10)=0.171-4.246*x34-1.313*x46
A(45,20)=1.183-4.937*x26
A(45,21)=3.399+2.468*x4*x3*cosx7+1.897*x4*x3*cosx47-
         0.352*x4*x23*cosx30-1.367*x4*x23*cosx34-2.493*
         x4*x27*cosx28
A(45,28)=3.465
A(45,37)=2.857
A(45,44)=0.205+0.564*x28+2.998*x32*x37
A(46,6)=2.274-4.104*x11-0.805*x37
A(46,17)=1.032
A(46,21)=0.556
A(46,24)=4.165+1.69*x24
A(46,36)=0.954+0.065*x9-0.429*x48*cosx21
A(47,14)=0.94+1.67*x45+3.897*x3*cosx38
A(47,20)=3.982
A(47,29)=2.81
A(48,17)=3.17
A(48,34)=3.816+2.848*x1
A(48,48)=3.446
b(1)=0.001*x27+0.002*x33+155.092*x40+0.001*x45
b(2)=0.006*x8+0.003*x14+0.315*u13+0.315*u14
b(4)=21.515*x40
b(5)=0.003*x8+0.007*x14+0.011*x26+0.001*x30+0.011*x32+
     0.004*x38+0.001*x42+0.001*x48+0.681*u13+0.681*u14
b(6)=0.002*x18+0.003*x24+0.003*x36+0.002*x42+0.002*x48+
     0.175*u13+0.175*u14
b(7)=0.001*x25+27.938*x32+0.001*x37+0.001*x43
b(8)=1.093*x16+0.003*x24+0.006*x30+0.006*x42+0.006*x48+
     0.017*u13+0.017*u14
b(9)=0.003*x15+0.002*x19+0.007*x21+0.003*x25+65.014*x28+
     87.078*x32+0.007*x33+0.003*x37+0.003*x39+0.003*x43+
     0.003*x45+0.345*u13+0.345*u14
b(10)=0.004*x7+0.009*x13+13.910*x20+0.003*x21+0.009*x25+
      0.002*x26+0.006*x27+0.004*x31+0.004*x32+151.368*
     x34+0.002*x38+0.006*x39+0.006*x45+u13+0.886*u14
b(11)=0.009*x10+0.012*x16+0.010*x28+0.001*x30+0.005*x34+
      0.001*x42+0.001*x48+1.154*u13+1.154*u14
b(12)=0.002*x10+0.002*x22+0.001*x25+0.002*x31+0.001*x37+
      326.416*x44+0.423*u13+0.423*u14
b(13)=0.004*x8+0.007*x14+0.007*x26+0.001*x29+0.004*x32+
      37.333*x36+0.001*x41+0.001*x47+0.744*u13+0.744*u14
b(14)=0.003*x1+0.006*x7+94.756*x14+0.006*x19+0.003*x25+
      3.017*u13+1.508*u14
b(16)=0.002*x25+0.004*x31+0.002*x37+594.201*x44
b(17)=0.004*x2+1.164*x6+0.009*x8+0.003*x9+0.002*x11+
      0.006*x15+0.009*x20+8.828*x22+0.004*x26+0.006*x27+
      0.003*x33+2.833*u13+5.011*u14
b(18)=0.002*x12+0.003*x18+0.003*x27+0.003*x30+0.007*x33+
       0.002*x36+408.861*x40+0.003*x45+0.339*u13+0.339*u14
b(19)=0.002*x11+31.150*x18+0.002*x22+0.002*x23+0.008*x28+
       0.008*x46+0.496*u13+0.496*u14
b(20)=4.560*x2+0.003*x3+0.007*x7+0.006*x9+0.004*x13+
       89.938*x16+0.006*x21+0.003*x25+0.003*x27+0.003*x30+
       0.006*x31+0.005*x36+0.003*x37+999.919*x44+0.003*
       x48+1.796*u13+1.796*u14
b(21)=0.004*x26+0.008*x32+0.004*x44+0.048*u13+0.048*u14
b(22)=0.003*x1+3.887*x6+0.006*x7+0.006*x11+100.685*x14+
       0.003*x17+0.006*x19+0.003*x25+3.516*u13+1.914*u14
b(23)=0.090*x6+0.002*x26+0.004*x30+0.004*x32+0.008*x36+
       0.002*x44+0.004*x48+0.007*u13+0.007*u14
b(25)=0.004*x24+0.007*x30+0.007*x42+0.007*x48
b(26)=0.008*x16+0.001*x22+0.003*x24+0.006*x30+0.006*x42+
       0.006*x48+0.786*u13+0.786*u14
b(27)=0.014*u13+0.014*u14
b(28)=1.384*x2+0.009*x5+0.002*x6+55.368*x12+0.001*x13+
       0.007*x14+0.003*x20+0.007*x23+0.005*x24+0.004*x26+
       0.005*x29+0.005*x30+0.009*x32+115.939*x36+0.005*
       x41+0.003*x42+0.004*x44+2.777*u13+2.777*u14
b(29)=0.010*x10+0.005*x16+9.380*x18+0.003*x22+0.008*x28+
       0.005*x29+0.009*x35+0.008*x40+0.004*x41+1350.946*
       x48+0.631*u13+0.631*u14
b(30)=0.009*x8+0.005*x14+0.002*x29+0.003*x35+0.002*x41+
       532.833*x48+0.452*u13+0.452*u14
b(31)=28.109*x8+0.010*x12+0.004*x13+0.003*x14+0.003*x16+
       0.005*x18+0.002*x19+0.007*x22+0.002*x24+0.004*x30+
       0.007*x34+0.003*x40+0.004*x42+0.003*x46+0.004*x48+
       1.529*u13+1.529*u14
b(32)=1.766*x4+0.003*x9+0.003*x10+0.001*x15+0.002*x16+
       0.359*u13+0.359*u14
b(34)=2.091*x4+0.010*x9+112.049*x16+0.007*x21+0.004*x27+
       0.003*x29+0.007*x35+0.003*x41+1045.660*x48+1.951*
       u13+1.951*u14
b(35)=0.002*x16+0.002*x17+0.005*x22+0.004*x23+39.585*x30+
       0.005*x34+0.004*x35+0.003*x36+0.002*x40+0.002*x41+
       0.001*x42+0.002*x46+0.443*u13+0.443*u14
b(36)=4.618*x2+2.092*x6+0.007*x7+0.003*x11+0.004*x13+
       0.005*x14+0.002*x17+0.004*x24+0.008*x30+0.014*x32+
       0.006*x38+0.008*x42+0.008*x48+1.009*u13+1.009*u14
b(37)=0.009*x6+0.009*x18+0.004*x24+0.005*x27+0.002*x30+
       0.009*x33+0.004*x36+0.005*x39+1468.843*x46+0.002*
       x48+0.894*u13+0.894*u14
b(38)=0.001*x26+0.001*x38+0.001*x44+0.128*u13+0.128*u14
b(39)=0.005*x1+0.002*x4+30.520*x8+0.003*x10+0.005*x13+
       0.002*x19+0.003*x22+0.002*x28+1.312*u13+1.312*u14
b(40)=0.003*x1+0.006*x7+0.004*x11+0.005*x12+88.862*x14+
       0.013*x17+0.003*x18+0.006*x19+12.759*x24+0.003*x25+
       89.546*x30+0.014*x35+0.005*x41+0.005*x47+4.385*u13+
       2.971*u14
b(41)=0.001*x23+0.002*x26+0.003*x29+0.004*x32+74.153*x36+
       0.003*x41+0.002*x44+0.003*x47
b(42)=0.004*x17+0.009*x23+0.002*x28+80.692*x30+0.005*x34+
       0.009*x35+0.004*x36+0.002*x40+0.004*x47+0.002*x48+
       0.428*u13+0.428*u14
b(43)=0.002*x19+0.003*x21+0.004*x25+0.006*x27+110.266*
       x32+161.192*x34+0.004*x37+0.006*x39+0.004*x43+
       0.006*x45
b(44)=0.004*x1+0.007*x7+0.004*x10+117.756*x14+0.007*x16
       0.005*x18+0.007*x19+0.009*x24+0.004*x25+0.007*x28+
       0.004*x34+0.009*x36+0.005*x42+0.005*x48+4.917*u13+
       3.042*u14
b(45)=0.010*x8+0.007*x14+0.007*x15+0.004*x16+10.680*x22+
       0.003*x25+0.003*x26+0.006*x31+0.002*x32+0.003*x33+
       0.007*x34+359.009*x38+0.003*x40+0.003*x46+1.758*
       u13+1.758*u14
b(46)=0.002*x11+0.009*x12+0.001*x15+32.438*x18+1.746*x22+
       0.001*x27+0.001*x29+0.010*x30+0.002*x42+0.002*x48+
      1.688*u13+1.688*u14
b(47)=0.006*x8+0.003*x17+0.006*x23+0.009*x26+52.948*x30+
      0.004*x32+0.006*x35+0.003*x41+0.003*x47+1.548*u13+
      2.018*u14
b(48)=0.003*x5+0.006*x11+99.562*x18+0.004*x22+0.006*x23+
      0.008*x28+0.003*x29+0.003*x30+0.007*x36+0.008*x40+
      0.003*x42+0.008*x46+1.585*u13+1.585*u14
```

benchmark #2

This ODE system is characterized by:

1. 95% sparsity for the matrix A distributed exponentially over its rows from 92% to 98%.

2. 99% sparsity for the matrices $\alpha, \beta, \gamma, \delta$ and θ of each nonzero element of A.

3. Linear terms as large as the nonlinear terms in the nonzero elements of A.

4. 50 Hz highest frequency.

```
A(1,39)=1.235+1.532*x48
A(2,2)=3.152
A(4,39)=0.171+4.660*x17
A(5,20)=3.406+4.246*x36
A(5,36)=0.611
A(5,44)=3.708+0.621*x14*x10-2.499*x41*x5*cosx37
A(6,30)=1.750-4.032*x36
A(7,10)=3.489
A(8,15)=0.035
A(8,36)=2.909+0.236*x35
A(9,27)=3.451-2.108*x7-3.349*x34
A(9,31)=1.733
A(10,19)=4.430
A(10,28)=1.298+2.706*x3-0.877*x21-0.477*x29+2.098*x38*
          x27+0.963*x16*x30*cosx3-1.355*x16*x30*cosx42
A(10,41)=2.141+2.627*x27*x30-0.265*x45*x8
A(11,1)=2.024-4.834*x26*cosx22
A(11,19)=4.759
A(11,33)=0.719+1.571*x9
A(12,13)=0.845
A(12,40)=1.039-0.824*x9*x2
A(13,17)=3.722+2.870*x40*x39*cosx27-0.921*x40*x39*cosx48
A(13,32)=0.743
A(14,6)=1.293-2.635*x2
A(14,19)=0.713
A(14,21)=4.782
A(15,8)=4.356-0.723*x32+4.646*x12*x29
A(15,15)=2.812-3.276*x26+4.908*x44
A(16,1)=2.957+3.233*x36+1.906*x32*x7-0.585*x32*x16
A(16,5)=0.066
A(16,9)=1.428+3.215*x34*cosx45
A(16,33)=1.736+3.853*x7
A(16,39)=4.617-3.887*x26+2.108*x25*cosx37
```

```
A(16,48)=3.648
A(17,14)=2.863-3.033*x42*x14
A(17,41)=2.698-0.467*x39-2.941*x16*x27
A(17,42)=3.183
A(18,3)=0.484
A(18,21)=3.405+2.297*x14+4.862*x15-2.770*x20+2.352*x46
A(18,34)=0.892-1.441*x13
A(18,35)=2.325+2.063*x48
A(18,48)=1.539
A(19,33)=1.976
A(19,37)=3.950-0.750*x37
A(21,31)=3.745-0.939*x21+2.378*x27-2.687*x12*x9-0.895*
         x12*x34
A(21,47)=4.984-2.400*x3-1.904*x17
A(22,5)=1.436
A(22,31)=3.181+2.841*x7+4.938*x29*x25
A(23,1)=0.138-0.610*x13-0.132*x28+3.768*x30+1.429*x48
A(23,5)=0.548-3.254*x37+3.895*x13*x4+2.406*x38*cosx8
A(23,29)=2.252-2.380*x5+1.257*x45
A(24,11)=2.648+2.799*x32*cosx41
A(24,21)=3.498
A(24,25)=2.307+0.546*x47*x48
A(24,28)=4.365
A(24,38)=2.909-1.828*x30+3.637*x22*x38
A(24,42)=4.814+3.860*x37
A(24,47)=2.988
A(25,2)=0.528
A(25,5)=0.347
A(25,16)=4.610-2.118*x11+4.668*x33*cosx18
A(25,17)=1.130-0.697*x32-3.221*x38
A(25,22)=0.279
A(26,5)=4.285
A(26,9)=0.422+2.178*x42
A(26,16)=3.012+1.229*x30*x27
A(26,19)=3.467
A(26,27)=1.832-0.078*x20+3.199*x19*cosx22
A(26,41)=0.432
A(26,42)=1.447
A(26,43)=1.706-1.245*x13
A(28,31)=0.223-2.438*x12+3.477*x26*cosx29-1.035*x26*
         cosx37
A(28,42)=1.673+0.353*x28*x29*cosx26
A(29,6)=3.567+2.849*x7
A(29,38)=3.329
A(30,1)=3.169-3.529*x25*cosx38
A(30,7)=0.974+1.878*x2*cosx44+4.075*x5*cosx39
A(30,10)=0.372
A(30,21)=0.217-2.522*x13
A(31,6)=2.655+0.373*x6+2.685*x42*x9-0.954*x42*x15
A(31,11)=0.811-3.323*x19
A(31,17)=3.101
A(31,27)=4.217-2.037*x38*x14
A(31,35)=3.321+4.356*x14
A(32,3)=4.472+1.389*x35*x11
A(32,23)=2.018
A(32,36)=4.676-4.546*x12
A(33,9)=0.255
A(33,23)=0.531
A(33,46)=2.430
A(34,7)=1.651
A(34,23)=0.490
A(34,27)=2.700
A(34,39)=4.774
A(35,26)=1.118-2.729*x44
A(35,35)=4.831
A(37,26)=1.475-1.682*x11
A(37,29)=1.928-1.049*x8*x3
A(37,30)=0.765
```

```
A(37,32)=3.025
A(38,37)=2.267-0.412*x29-2.633*x12*cosx10
A(39,1)=1.818+3.991*x47
A(39,20)=4.936-3.827*x22
A(39,22)=0.426+4.595*x28
A(39,23)=0.147-2.275*x15-3.151*x44*x18-4.166*x44*x26+
         4.714*x44*x39
A(39,30)=4.528+0.941*x48-2.516*x30*x2-3.903*x30*x18+
         0.525*x42*x17
A(39,33)=2.304-0.911*x17
A(39,39)=4.611
A(39,47)=0.171-4.246*x34-1.313*x46
A(40,9)=1.183-4.937*x26
A(40,10)=3.399+2.468*x4*x3*cosx7+1.897*x4*x3*cosx47-
         0.352*x4*x23*cosx30-1.367*x4*x23*cosx34-2.493*
         x4*x27*cosx28
A(40,17)=3.465
A(40,26)=2.857
A(40,33)=0.205+0.564*x28+2.998*x32*x37
A(40,43)=2.274-4.104*x11-0.805*x37
A(41,6)=1.032
A(41,10)=0.556
A(41,13)=4.165+1.690*x24
A(41,25)=0.954+0.065*x9-0.429*x48*cosx21
A(42,3)=0.940+1.670*x45+3.897*x3*cosx38
A(42,9)=3.982
A(42,18)=2.810
A(43,6)=3.170
A(43,23)=3.816+2.848*x1
A(43,37)=3.446
A(44,10)=3.818-4.173*x13+4.299*x41*cosx47
A(44,21)=4.557-2.634*x46
A(44,42)=4.933
A(44,43)=3.062-2.008*x2-3.488*x46*cosx36
A(45,15)=4.766
A(45,19)=3.761
A(45,26)=2.722-3.654*x36*cosx8+4.959*x36*cosx39
A(45,37)=0.098-1.086*x21*x46
A(46,9)=4.974-4.918*x27
A(46,11)=4.694+0.601*x25*cosx26
A(46,13)=4.444
A(46,30)=1.043+0.370*x28*x8
A(46,32)=2.648-0.318*x5*cosx2-4.551*x5*cosx9+4.039*x16*
         x19*cosx19+2.604*x16*x19*cosx42
A(46,40)=2.547+0.267*x42*x20*cosx35
A(46,45)=2.936-2.995*x46+1.641*x48-2.191*x31*x3-1.899*
         x35*cosx38
A(47,8)=3.410
A(47,13)=3.784+4.218*x28*x32*cosx5
A(47,15)=0.331
A(48,34)=1.756
A(48,43)=1.395-4.612*x5*cosx7
b(1)=0.001*x27+0.002*x33+155.092*x40+0.001*x45
b(2)=0.006*x8+0.003*x14+0.315*u13+0.315*u14
b(4)=21.515*x40
b(5)=0.003*x8+0.007*x14+0.011*x26+0.001*x30+0.011*x32+
     0.004*x38+0.001*x42+0.001*x48+ 0.681*u13+0.681*u14
b(6)=0.002*x18+0.003*x24+0.003*x36+0.002*x42+0.002*x48+
     0.175*u13+0.175*u14
b(7)=0.007*x4+0.007*x16+0.003*x22+0.698*u13+0.698*u14
b(8)=1.093*x16+0.003*x24+0.006*x30+0.006*x42+0.006*x48+
     0.017*u13+0.017*u14
b(9)=0.003*x15+0.002*x19+0.007*x21+0.003*x25+65.014*x28+
     87.078*x32+0.007*x33+0.003*x37+0.003*x39+0.003*x43+
     0.003*x45+0.345*u13+0.345*u14
b(10)=0.004*x7+0.009*x13+0.001*x16+13.910*x20+0.003*x22+
      0.009*x25+0.002*x29+0.004*x31+0.003*x34+0.004*x35+
      0.001*x40+269.094*x42+0.001*x46+0.002*x47+1.016*
```

```
        u13+1.016*u14
b(11)=2.530*x2+0.009*x7+0.012*x13+14.943*x20+0.010*x25+
        0.001*x27+0.005*x31+36.161*x34+0.001*x39+0.001*x45+
        1.154*u13+1.154*u14
b(12)=0.002*x7+26.548*x14+0.002*x19+0.001*x28+0.002*x34+
        0.001*x46
b(13)=0.004*x5+0.007*x11+116.910*x18+0.007*x23+0.001*x26+
        0.004*x29+0.001*x38+0.001*x44+1.861*u13+1.861*u14
b(14)=0.005*x9+0.003*x12+0.001*x13+0.010*x15+0.001*x18+
        2.239*x20+15.014*x22+0.001*x25+0.010*x27+0.005*
        x33+1.228*u13+1.228*u14
b(15)=0.009*x2+0.003*x3+0.006*x9+0.009*x14+88.310*x16+
        0.004*x20+0.006*x21+0.003*x27+2.277*u13+2.277*u14
b(16)=3.696*x2+0.003*x3+0.083*x6+0.006*x7+17.930*x10+
        0.003*x13+0.003*x15+0.003*x21+0.008*x27+0.004*x30+
        87.243*x34+0.007*x36+580.119*x40+0.004*x42+0.008*
        x45+0.588*u13+0.588*u14
b(17)=0.003*x2+0.006*x8+0.006*x20+0.003*x26+0.003*x29+
        0.003*x30+0.005*x35+0.006*x36+338.915*x42+0.003*
        x47+0.003*x48+1.432*u13+2.863*u14
b(18)=0.605*x4+0.004*x9+0.007*x15+10.692*x22+0.002*x23+
        0.007*x27+0.002*x28+0.005*x29+0.002*x30+116.809*
        x36+0.002*x40+0.005*x41+0.002*x42+0.002*x46+0.729*
        u13+0.729*u14
b(19)=0.002*x21+0.004*x25+0.004*x27+0.008*x31+99.296*x34+
        496.356*x38+0.004*x39+0.004*x43+0.004*x45
b(21)=0.004*x19+0.007*x25+0.005*x29+188.211*x32+0.010*
        x35+0.007*x37+0.005*x41+0.007*x43+1565.722*x48
b(22)=1.796*x6+0.003*x11+0.001*x17+0.003*x19+0.006*x25+
        159.862*x32+0.006*x37+0.006*x43+0.144*u13+0.144*u14
b(23)=0.172*x2+0.685*x6+0.001*x11+0.003*x17+0.005*x23+
        42.426*x30+0.005*x35+0.002*x41+0.002*x47+0.294*u13+
        0.294*u14
b(24)=0.005*x5+0.003*x9+33.258*x12+0.002*x13+0.007*x15+
        0.004*x16+0.005*x17+0.005*x19+10.993*x22+0.003*x23+
        43.461*x26+0.003*x29+0.005*x30+0.005*x31+0.006*x32+
        0.003*x33+0.009*x34+0.006*x35+0.010*x36+0.004*x40+
        0.002*x43+0.003*x44+0.004*x46+938.620*x48+1.896*
        u13+1.896*u14
b(25)=0.005*x4+0.434*x6+0.001*x8+0.009*x10+0.003*x11+
        35.508*x18+0.002*x22+0.002*x23+0.005*x28+0.001*x29+
        3.014*u13+3.014*u14
b(26)=5.356*x6+0.003*x7+5.305*x10+0.009*x11+0.007*x13+
        0.004*x17+10.887*x20+0.004*x21+0.009*x25+34.523*
        x28+0.001*x30+0.007*x31+0.004*x33+0.003*x36+0.002*
        x37+0.002*x39+54.224*x42+536.020*x44+0.002*x45+
        0.001*x48+1.608*u13+1.608*u14
b(28)=0.002*x30+11.194*x32+0.003*x36+0.002*x48
b(29)=0.007*x12+0.004*x18+0.003*x26+0.007*x32+0.003*x44+
        0.357*u13+0.357*u14
b(30)=3.962*x2+12.230*x8+0.005*x13+0.681*x22+0.629*u13+
        0.629*u14
b(31)=10.196*x12+0.004*x15+97.407*x18+0.008*x21+0.010*
        x23+79.445*x28+0.010*x29+0.008*x33+166.930*x36+
        0.004*x39+0.007*x41+0.004*x45+0.007*x47+2.400*u13+2.400*u14
b(32)=5.591*x4+0.009*x9+0.002*x11+0.004*x15+0.004*x17+
        6.341*x24+0.004*x29+0.009*x30+0.009*x42+0.009*x48+
        0.851*u13+0.851*u14
b(33)=3.204*x10+0.001*x17+1.667*x24+0.002*x28+0.001*x29+
        0.005*x34+0.002*x40+0.157*u13+0.157*u14
b(34)=0.003*x1+20.740*x8+0.003*x13+0.003*x15+0.002*x19+
        0.005*x21+1.540*x24+50.876*x28+0.015*x33+599.877*
        x40+0.007*x45+0.698*u13+0.698*u14
b(35)=0.001*x14+0.002*x20+0.005*x23+0.010*x29+0.002*x32+
        242.785*x36+0.001*x38+0.010*x41+0.001*x44+0.010*
        x47+0.112*u13+0.112*u14
b(37)=0.001*x14+0.002*x17+0.006*x20+0.004*x23+0.002*x24+
        36.311*x30+0.004*x35+0.002*x36+0.008*x38+0.002*x41+
        0.008*x44+0.002*x47+0.417*u13+0.417*u14
b(38)=0.002*x25+0.005*x31+284.822*x38+0.002*x43
b(39)=2.272*x2+0.004*x7+0.005*x8+0.002*x13+0.010*x14+
        0.005*x18+0.471*x24+0.010*x26+0.009*x27+0.005*x32+
        115.785*x34+0.009*x36+579.398*x40+0.005*x42+0.009*
        x45+53.612*x48+1.736*u13+1.736*u14
b(40)=0.002*x3+0.007*x4+0.003*x5+14.774*x10+0.007*x11+
        0.003*x14+0.002*x15+0.007*x16+108.842*x18+0.006*
        x20+0.001*x21+0.003*x22+0.007*x23+0.003*x29+0.005*
        x31+0.006*x32+10.317*x34+0.002*x37+0.003*x38+
        714.470*x44+2.935*u13+2.935*u14
b(41)=0.004*x1+0.001*x4+0.008*x7+0.002*x12+130.826*x14+
        0.001*x16+0.001*x18+0.010*x19+17.971*x26+0.002*x31+
        4.475*u13+2.392*u14
b(42)=1.175*x4+0.003*x6+50.018*x10+0.006*x12+0.009*x15+
        0.004*x21+0.006*x24+0.003*x30+2.296*u13+2.296*u14
b(43)=0.004*x11+0.006*x12+0.008*x17+0.003*x18+11.981*x24+
        0.003*x25+0.008*x29+0.007*x31+0.004*x35+433.076*
        x38+0.003*x43+1.080*u13+1.080*u14
b(44)=0.008*x4+0.009*x15+0.008*x16+14.312*x22+0.003*x25+
        0.009*x27+0.005*x30+0.006*x31+0.005*x33+0.010*x36+
        0.003*x37+962.050*x44+0.005*x48+1.675*u13+1.675*u14
b(45)=0.005*x3+0.004*x7+0.010*x9+0.008*x13+0.003*x14+
        149.710*x16+11.816*x20+0.010*x21+0.005*x27+0.004*
        x31+0.005*x32+12.322*x38+0.003*x44+3.408*u13+3.408*
        u14
b(46)=0.004*x1+0.010*x3+0.009*x5+0.009*x7+62.478*x10+
        58.960*x12+139.582*x14+0.010*x15+0.009*x17+0.001*
        x18+0.009*x19+0.003*x20+0.005*x21+0.005*x23+0.002*
        x24+0.004*x25+0.005*x26+0.003*x27+0.003*x28+0.006*
        x33+0.005*x34+0.002*x36+0.005*x38+0.001*x42+0.005*
        x44+922.171*x46+0.001*x48+6.482*u13+4.260*u14
b(47)=0.004*x1+0.007*x2+118.862*x14+10.386*x16+0.008*x19+
        0.003*x20+0.004*x25+4.631*u13+2.739*u14
b(48)=0.002*x22+0.001*x25+0.004*x28+0.003*x31+0.001*x37+
        0.004*x40+438.354*x44+0.004*x46
```

benchmark #3

This ODE system is characterized by:

1. 95% sparsity for the matrix A distributed inverse exponentially over its rows from 92% to 98%.

2. 99% sparsity for the matrices $\alpha, \beta, \gamma, \delta$ and θ of each nonzero element of A.

3. Linear terms as large as the nonlinear terms in the nonzero elements of A.

4. 50 Hz highest frequency.

```
A(1,39)=1.235+1.532*x48
A(2,2)=3.152
A(5,4)=4.253+4.660*x4
A(6,10)=2.014
```

```
A(6,43)=2.068+1.749*x4*cosx1+0.621*x18*cosx10
A(7,19)=0.209-1.498*x19+3.208*x21-2.938*x8*cosx38
A(7,39)=2.125-2.019*x20
A(10,14)=4.808
A(10,32)=4.532+0.125*x47*x37
A(10,35)=2.637-4.085*x42*cosx30
A(11,39)=0.470+3.862*x37
A(13,44)=0.397-1.517*x33*x21*cosx11+2.098*x33*x42*cosx27
A(13,48)=3.121+3.020*x47*x30+0.963*x47*x31-2.776*x25*x30*
         cosx38-0.265*x25*x30*cosx44
A(14,35)=2.336
A(14,36)=3.225-4.834*x42*x22
A(14,43)=4.667
A(14,48)=3.851+2.991*x12
A(16,7)=2.954-0.824*x47*cosx2
A(18,22)=3.722+2.870*x40*x39*cosx27-0.921*x40*x39*cosx48
A(18,37)=0.743
A(19,15)=3.017
A(21,45)=1.891
A(22,7)=0.931
A(22,16)=4.356-0.723*x32+4.646*x12*x29
A(23,3)=1.167+4.908*x19-1.139*x40
A(23,15)=1.627+4.839*x30*x47-0.585*x34*x11
A(23,16)=0.066
A(23,20)=1.428+3.215*x34*cosx45
A(24,5)=2.887
A(24,13)=2.444-3.887*x18+2.108*x17*cosx37
A(24,14)=3.648
A(24,28)=2.863-3.033*x42*x14
A(25,7)=2.698-0.467*x39-2.941*x16*x27
A(25,8)=3.183
A(25,47)=1.134
A(26,3)=3.971+4.862*x2-2.770*x7+2.352*x33
A(26,18)=3.109
A(26,26)=3.205+2.063*x29
A(27,18)=0.072+3.105*x7*cosx42
A(28,3)=1.976
A(28,7)=3.950-0.750*x37
A(30,1)=3.745-0.939*x21+2.378*x27-2.687*x12*x9-0.895*x12*
        x34
A(30,17)=4.984-2.400*x3-1.904*x17
A(31,19)=3.696
A(31,35)=3.135+4.676*x23
A(31,36)=3.455-0.132*x16+3.768*x18+1.429*x36+4.686*x46
A(31,44)=2.625-3.747*x18*x27+3.895*x18*x29
A(32,4)=2.252-2.380*x5+1.257*x45
A(32,45)=0.835+2.799*x21*cosx41
A(33,22)=0.556
A(33,29)=4.941+2.084*x26+0.546*x15*cosx48
A(33,34)=2.955
A(34,10)=2.909-1.828*x30+3.637*x22*x38
A(34,14)=4.814+3.860*x37
A(34,27)=0.299
A(34,44)=3.126+3.758*x47
A(35,22)=0.347
A(35,33)=4.610-2.118*x11+4.668*x33*cosx18
A(35,34)=1.130-0.697*x32-3.221*x38
A(35,39)=0.279
A(36,22)=4.285
A(37,16)=3.619+2.178*x4+3.885*x34
A(37,22)=2.497+1.229*x38*x27
A(37,33)=3.467
A(37,41)=1.832-0.078*x20+3.199*x19*cosx22
A(38,7)=0.432
A(38,8)=1.447
A(38,9)=1.706-1.245*x13
A(39,45)=0.223-2.438*x12+3.477*x26*cosx29-1.035*x26*
         cosx37
```

```
A(40,8)=1.673+0.353*x28*x29*cosx26
A(40,20)=3.567+2.849*x7
A(41,4)=3.329
A(41,33)=2.330-3.529*x7*cosx38
A(41,34)=2.101+1.878*x37*x44+4.075*x40*x39+0.771*x47*
         cosx42
A(42,5)=1.271
A(42,6)=3.695-3.595*x25-2.522*x37
A(42,10)=1.673-0.954*x2*x9*cosx8
A(42,25)=0.811-3.323*x19
A(42,31)=3.101
A(42,41)=4.217-2.037*x38*x14
A(43,1)=3.321+4.356*x14
A(43,17)=4.472+1.389*x35*x11
A(43,37)=2.018
A(44,2)=4.676-4.546*x12
A(44,23)=0.255
A(44,37)=0.531
A(45,12)=2.430
A(45,21)=1.651
A(45,37)=0.490
A(45,41)=2.700
A(46,5)=4.774
A(46,40)=1.118-2.729*x44
A(47,1)=4.831
A(48,40)=1.475-1.682*x11
A(48,43)=1.928-1.049*x8*x3
A(48,44)=0.765
A(48,46)=3.025
b(1)=0.001*x27+0.002*x33+155.092*x40+0.001*x45
b(2)=0.006*x8+0.003*x14+0.315*u13+0.315*u14
b(5)=0.009*x10+0.004*x16+0.425*u13+0.425*u14
b(6)=0.004*x4+0.004*x16+0.002*x22+0.002*x25+0.004*x31+
     0.002*x37+649.630*x44+0.403*u13+0.403*u14
b(7)=0.658*x20+0.002*x27+0.004*x33+266.985*x40+0.002*x45+
     0.042*u13+0.042*u14
b(10)=0.005*x2+0.010*x8+0.014*x20+0.003*x23+0.014*x26+
      0.005*x29+132.541*x36+0.009*x38+0.005*x41+0.009*
      x44+0.005*x47+2.404*u13+4.808*u14
b(11)=59.103*x40
b(13)=0.003*x30+0.006*x36+0.003*x42
b(14)=0.002*x23+0.003*x24+0.005*x25+0.005*x29+0.010*x30+
      0.009*x31+117.318*x36+0.005*x37+0.005*x41+0.010*
      x42+1466.134*x44
b(16)=0.006*x1+37.106*x8+0.006*x13+0.003*x19+0.591*u13+
      0.591*u14
b(18)=0.004*x10+0.007*x16+0.007*x28+0.001*x31+0.004*x34+
      93.341*x38+0.744*u13+0.744*u14
b(19)=0.003*x3+0.006*x9+94.756*x16+0.006*x21+0.003*x27+
      1.508*u13+1.508*u14
b(21)=0.002*x27+0.004*x33+0.002*x39+594.201*x46
b(22)=0.002*x1+0.004*x4+11.699*x8+0.009*x10+0.002*x13+
      0.009*x22+0.004*x28+2.364*u13+2.364*u14
b(23)=1.459*x4+0.001*x8+0.006*x9+0.003*x14+51.104*x16+
      0.003*x21+0.003*x26+0.002*x27+0.001*x32+1.249*u13+
      1.249*u14
b(24)=0.002*x1+0.004*x2+3.609*x6+0.005*x7+0.007*x8+0.006*
      x11+76.674*x14+0.003*x16+0.003*x17+0.005*x19+0.007*
      x20+0.006*x22+0.002*x25+0.004*x26+0.006*x34+0.003*
      x40+0.003*x46+4.843*u13+5.445*u14
b(25)=0.005*x1+0.006*x2+33.804*x8+0.005*x13+0.006*x14+
      0.003*x19+0.003*x20+0.001*x29+0.002*x35+0.001*x41+
      356.200*x48+1.176*u13+1.176*u14
b(26)=4.964*x4+0.003*x6+0.008*x9+0.006*x12+0.003*x14+
      0.004*x15+0.006*x20+0.006*x24+0.003*x30+0.006*x32+
      0.003*x38+0.003*x44+2.272*u13+2.272*u14
b(27)=0.036*u13+0.036*u14
b(28)=0.008*x1+2.470*x4+49.612*x8+0.004*x9+0.008*x13+
          0.002*x15+0.004*x19+0.988*u13+0.988*u14
b(30)=4.681*x2+0.005*x5+0.007*x7+0.010*x11+0.004*x13+
          156.547*x18+0.010*x23+0.005*x29+2.866*u13+2.866*u14
b(31)=0.004*x7+0.007*x13+11.606*x20+0.003*x23+0.003*x24+
          0.007*x25+0.003*x26+0.006*x29+0.007*x30+0.004*x31+
          0.005*x32+157.475*x36+0.003*x38+0.006*x41+0.007*
          x42+0.006*x47+0.007*x48+0.739*u13+0.739*u14
b(32)=0.005*x10+0.002*x16+0.002*x33+262.271*x46+0.225*
          u13+0.225*u14
b(33)=0.001*x16+0.005*x17+0.010*x23+0.007*x28+93.092*x30+
          0.010*x35+0.006*x40+0.005*x41+0.006*x46+0.005*x47+
          0.605*u13+0.605*u14
b(34)=0.005*x2+0.006*x4+0.010*x8+0.006*x16+0.010*x20+
          0.003*x22+0.008*x26+5.626*x28+0.006*x32+0.003*x38+
          3.019*u13+5.425*u14
b(35)=0.009*x27+0.003*x28+231.683*x34+0.002*x39+35.027*
          x40+0.009*x45+0.002*x46+0.069*u13+0.069*u14
b(36)=0.004*x10+0.009*x16+0.009*x28+0.004*x34+0.857*u13+
          0.857*u14
b(37)=0.004*x4+0.010*x10+0.007*x27+0.009*x28+0.002*x29+
          174.270*x34+0.004*x35+0.007*x39+230.257*x42+0.007*
          x45+0.002*x47+2.309*u13+2.309*u14
b(38)=0.003*x2+0.003*x3+5.387*x8+21.431*x10+0.003*x14+
          0.003*x15+0.001*x20+0.002*x21+0.717*u13+0.717*u14
b(39)=69.967*x46
b(40)=0.003*x2+0.010*x14+0.007*x26+0.004*x32+1.048*u13+
          1.048*u14
b(41)=0.007*x10+0.003*x16+0.002*x21+0.002*x22+0.005*x27+
          0.004*x28+117.067*x34+0.005*x39+0.004*x40+0.005*
          x45+0.004*x46+0.333*u13+0.333*u14
b(42)=0.003*x4+1.496*x6+0.003*x11+0.007*x12+0.003*x16+
          0.001*x17+0.004*x18+0.005*x19+0.002*x22+15.286*x26+
          0.004*x29+155.860*x32+0.008*x35+0.007*x37+529.890*
          x42+0.007*x43+0.004*x47+0.912*u13+0.912*u14
b(43)=4.152*x2+0.004*x5+0.007*x7+0.009*x11+0.003*x13+
          140.480*x18+0.009*x23+0.002*x25+0.004*x29+0.004*
          x31+253.590*x38+0.002*x43+2.568*u13+2.568*u14
b(44)=0.009*x8+0.005*x14+0.801*x24+0.001*x31+66.698*x38+
          0.519*u13+0.519*u14
b(45)=0.005*x6+0.002*x9+0.003*x15+0.005*x18+5.185*x22+
          0.002*x24+0.003*x27+0.003*x29+0.002*x33+0.005*x35+
          61.632*x38+339.333*x42+0.003*x47+0.816*u13+0.816*
          u14
b(46)=5.967*x6+0.010*x11+0.005*x17+0.001*x28+0.002*x34+
          0.001*x46+0.477*u13+0.477*u14
b(47)=6.038*x2+0.010*x7+0.005*x13+0.483*u13+0.483*u14
b(48)=0.002*x25+0.005*x28+0.004*x31+0.002*x32+0.009*x34+
          0.002*x37+605.766*x44
```

benchmark #4

This ODE system is characterized by:

1. 95% sparsity for the matrix A distributed linearly over its rows from 92% to 98% with the last row being long with 0% sparsity.

2. 99% sparsity for the matrices $\alpha, \beta, \gamma, \delta$ and θ of each nonzero element of A, with 90% sparsity for the elements of the last row.

3. Linear terms as large as the nonlinear terms in the nonzero elements of A.

4. 50 Hz highest frequency.

```
A(1,39)=1.235+1.532*x48
A(2,2)=3.152
A(5,4)=4.253+4.660*x4
A(6,10)=2.014
A(6,43)=2.068+1.749*x4*cosx1+0.621*x18*cosx10
A(7,19)=0.209-1.498*x19+3.208*x21-2.938*x8*cosx38
A(7,39)=2.125-2.019*x20
A(10,14)=4.808
A(10,32)=4.532+0.125*x47*x37
A(10,35)=2.637-4.085*x42*cosx30
A(11,39)=0.470+3.862*x37
A(13,44)=0.397-1.517*x33*x21*cosx11+2.098*x33*x42*cosx27
A(13,48)=3.121+3.020*x47*x30+0.963*x47*x31-2.776*x25*x30*
         cosx38-0.265*x25*x30*cosx44
A(14,35)=2.336
A(14,36)=3.225-4.834*x42*x22
A(14,43)=4.667
A(14,48)=3.851+2.991*x12
A(16,7)=2.954-0.824*x47*cosx2
A(18,22)=3.722+2.870*x40*x39*cosx27-0.921*x40*x39*cosx48
A(18,42)=0.678
A(19,15)=3.017
A(21,45)=1.891
A(22,7)=0.931
A(22,16)=4.356-0.723*x32+4.646*x12*x29
A(23,3)=1.167+4.908*x19-1.139*x40
A(24,26)=1.693+3.233*x19+1.906*x15*x7-0.585*x15*x16
A(24,41)=3.254
A(25,19)=0.992-0.346*x32+4.125*x26*x23-1.179*x26*x32
A(25,45)=2.887
A(26,25)=4.539+2.795*x42+2.108*x45*cosx37
A(27,6)=3.648
A(27,37)=1.850-3.033*x25*x14
A(28,40)=1.626-0.459*x26
A(28,42)=0.445-1.709*x17
A(29,38)=1.134
A(29,42)=3.971+4.862*x2-2.770*x7+2.352*x33
A(30,9)=3.109
A(30,17)=3.205+2.063*x29
A(31,9)=0.072+3.105*x7*cosx42
A(31,42)=1.976
A(31,46)=3.950-0.750*x37
A(33,40)=3.745-0.939*x21+2.378*x27-2.687*x12*x9-0.895*
         x12*x34
A(34,8)=4.984-2.400*x3-1.904*x17
A(35,10)=3.696
A(35,26)=3.135+4.676*x23
A(35,27)=3.455-0.132*x16+3.768*x18+1.429*x36+4.686*x46
A(37,7)=1.102-1.794*x25+2.724*x26+2.406*x46*cosx8
A(37,14)=3.277+1.005*x30*x43
A(37,17)=4.408
A(37,24)=2.259
A(38,22)=4.941+2.084*x26+0.546*x15*cosx48
A(38,27)=2.955
A(39,3)=2.909-1.828*x30+3.637*x22*x38
A(39,24)=4.503+3.860*x20
A(39,43)=1.608
A(39,47)=4.058+3.758*x37
A(40,15)=0.347
```

```
A(40,26)=4.610-2.118*x11+4.668*x33*cosx18
A(40,27)=1.130-0.697*x32-3.221*x38
A(40,32)=0.279
A(43,11)=0.026+2.596*x35
A(43,12)=2.728+3.885*x31
A(43,48)=0.249+1.229*x8*x27
A(44,9)=4.562
A(44,11)=3.670+3.199*x48*x22
A(44,12)=3.484
A(45,48)=4.470+2.253*x27
A(46,9)=3.220
A(46,27)=1.588+2.143*x27-0.251*x34
A(47,19)=3.133-2.011*x17-4.068*x43
A(48,1)=2.834-1.569*x11*x30-3.323*x11*x42
A(48,2)=4.336-3.529*x35*cosx38+2.689*x35*cosx40+0.925*
        x2*x22*cosx44
A(48,3)=1.828-1.663*x13+2.577*x28-3.701*x42+4.199*x48+
        4.075*x6*x39-0.913*x21*x9+0.247*x21*x38+0.771*
        x29*x22+1.967*x29*x46+4.808*x45*cosx42-4.605*x48*
        x44*cosx22-2.073*x48*x44*cosx35-1.237*x48*x44*
        cosx39
A(48,4)=1.844+0.223*x1-3.645*x15+1.275*x46-0.217*x10*x27-
        2.726*x10*x40-0.682*x13*x12+3.721*x13*x28-4.882*
        x13*x48+1.831*x12*x10*cosx30+1.070*x12*x10*
        cosx42+1.046*x21*x11*cosx7-2.323*x21*x11*cosx41+
        2.685*x21*x35*cosx9+4.904*x21*x35*cosx12-0.954*
        x21*x35*cosx13+1.830*x21*x35*cosx15+1.011*x21*
        x35*cosx40-2.256*x39*x30*cosx13+1.590*x39*x42*
        cosx3
A(48,5)=3.217+2.944*x18+4.114*x35*x8-3.008*x35*x25+0.436*
        x35*x31-3.045*x48*cosx6+2.418*x48*cosx13-4.679*
        x48*cosx24
A(48,6)=2.184+2.610*x11+3.660*x25+3.001*x40-2.037*x42*
        x14-4.494*x42*x34+3.334*x36*cosx7-2.794*x36*
        cosx8+2.461*x36*cosx10-2.678*x36*cosx31-3.743*
        x36*cosx44-3.730*x23*x1*cosx25+1.680*x23*x23*
        cosx16+1.738*x23*x23*cosx40+0.683*x23*x25*cosx31-
        4.472*x44*x36*cosx29-1.158*x44*x36*cosx40
A(48,7)=0.626-2.640*x24-3.200*x40+2.272*x48-2.108*x10*
        x36-4.452*x10*x38+4.116*x10*x43+3.275*x19*x6+
        4.850*x38*x20+3.741*x38*x34
A(48,8)=1.527+0.491*x35*cosx38+0.763*x35*cosx46-2.656*
        x8*x43*cosx1+0.760*x8*x43*cosx4+2.212*x22*x23*
        cosx10-3.818*x22*x23*cosx19-2.009*x22*x23*cosx37
A(48,9)=1.284+4.330*x2+3.108*x33+1.051*x46
A(48,10)=1.436+0.715*x31+0.030*x32+4.257*x46+1.745*x44*
         x36-2.339*x48*cosx20-4.257*x48*cosx26-2.619*x48*
         cosx32-2.049*x16*x47*cosx4
A(48,11)=4.470-3.789*x12+2.724*x45*x19
A(48,12)=1.560-0.216*x29-2.164*x37-4.175*x43+0.782*x32*
         cosx22+3.260*x18*x35*cosx24+1.722*x31*x5*cosx12-
         4.376*x31*x9*cosx3+4.820*x31*x9*cosx38+4.002*
         x31*x43*cosx26
A(48,13)=2.126-2.729*x41+1.102*x6*cosx26-2.028*x20*x30*
         cosx46
A(48,14)=3.913-1.796*x5-3.512*x37*cosx13+3.053*x1*x1*
         cosx20
A(48,15)=2.285-1.641*x3-4.460*x35-3.006*x23*x19-1.049*
         x23*x20-3.619*x31*x43-0.752*x13*cosx17-4.283*x8*x11*cosx48
A(48,16)=1.824+2.995*x5+4.175*x20+0.023*x30*cosx34+0.227*
         x48*cosx2+0.512*x48*cosx43
A(48,17)=0.847+1.381*x33-0.412*x48+0.818*x3*cosx28-2.633*
         x3*cosx36+3.037*x3*cosx47+0.154*x15*x38*cosx23+
         4.677*x28*x35*cosx4+4.447*x28*x35*cosx47-4.658*
         x47*x2*cosx20-3.868*x47*x47*cosx14
A(48,18)=3.328+4.293*x27*x7+1.092*x27*x48-4.864*x17*x17*
         cosx13-1.282*x17*x21*cosx34
A(48,19)=3.434-2.275*x9+0.173*x34-3.151*x35*x18-3.078*
```

```
         x35*x23-4.166*x35*x24+4.714*x35*x37-4.278*x34*
         x10*cosx14+1.360*x34*x10*cosx16+2.168*x34*x10*
         cosx46-2.516*x34*x39*cosx2-2.591*x34*x39*cosx15-
         3.903*x34*x39*cosx16-3.408*x34*x39*cosx42+1.095*
         x34*x41*cosx9-3.453*x34*x41*cosx20-4.805*x34*
         x41*cosx37-4.181*x34*x41*cosx48+0.525*x34*x43*
         cosx13
A(48,20)=0.506+1.786*x21+4.719*x38-0.109*x25*x19-4.477*
         x29*cosx4-0.538*x29*cosx7-0.739*x4*x31*cosx1+
         3.475*x4*x31*cosx14-2.425*x45*x43*cosx9+1.222*
         x45*x43*cosx22-1.287*x45*x43*cosx48
A(48,21)=2.153+1.947*x6-2.683*x4*x8-2.582*x4*x21+3.808*
         x16*x14+3.284*x16*x3*cosx14
A(48,22)=0.557-2.979*x21+2.658*x29-0.465*x11*x26+2.767*
         x11*x27+2.468*x11*x32-3.168*x11*x40+1.798*x33*
         x3+0.966*x33*x20+4.719*x34*x7-4.990*x34*x14+
         0.821*x34*x17+3.700*x34*x33-0.352*x34*x40-1.367*
         x34*x44-2.493*x48*x28+3.465*x41*cosx31+4.281*x2*
         x13*cosx3+1.652*x47*x27*cosx2-2.409*x47*x27*
         cosx15
A(48,23)=0.644+3.793*x22*x15+2.998*x22*x35+3.364*x26*x3+
         1.182*x26*x40-0.939*x29*x31+1.129*x6*cosx5-
         3.490*x6*cosx19-2.343*x1*x48*cosx43
A(48,24)=3.057+0.638*x6+1.450*x47
A(48,25)=2.187-3.014*x33
A(48,26)=2.065+4.681*x13-0.278*x10*x19-2.672*x21*x5-
         3.254*x44*x1+3.046*x44*x38+1.121*x46*x10-0.817*
         x46*x41-0.429*x5*cosx21
A(48,27)=2.180-1.218*x26+1.495*x22*x39-2.117*x22*x42-
         3.986*x25*x8*cosx19+3.627*x25*x8*cosx21
A(48,28)=1.613-3.816*x25*x40*cosx30+4.679*x25*x40*cosx31-
         0.657*x25*x40*cosx47
A(48,31)=0.792-2.319*x25-2.634*x30-2.488*x7*x5*cosx18-
         3.813*x30*x37*cosx9+0.750*x30*x37*cosx21+3.062*
         x30*x37*cosx31-0.645*x30*x37*cosx33-4.020*x30*
         x42*cosx33
A(48,32)=0.990+4.939*x5+0.289*x36*x5-3.488*x36*x39+1.684*
         x13*x15*cosx48-4.891*x13*x18*cosx18-2.353*x13*
         x40*cosx26
A(48,33)=4.666+4.762*x43-2.722*x30*cosx34-0.922*x30*
         cosx45-3.193*x1*x27*cosx22-3.654*x1*x27*cosx28+
         2.661*x1*x27*cosx44+3.073*x1*x36*cosx2-4.150*x1*
         x41*cosx31+2.860*x1*x41*cosx39
A(48,34)=0.150-4.611*x2+1.508*x34-1.906*x29*x9-3.533*x36*
         x3-1.059*x30*cosx27+2.516*x30*cosx28-3.286*x30*
         cosx31-1.064*x42*cosx44+4.927*x19*x42*cosx4+
         1.839*x19*x42*cosx7-4.694*x19*x42*cosx10-1.077*
         x19*x42*cosx37+4.931*x19*x42*cosx46
A(48,35)=1.632+3.381*x41+1.217*x46-1.774*x13*x11+0.309*
         x13*x36-1.464*x9*x22*cosx36
A(48,36)=4.624+3.833*x17-0.071*x34-1.875*x28*x14+2.954*
         x16*cosx4+3.361*x23*cosx1+3.380*x23*cosx16-
         3.812*x23*cosx17+0.370*x23*cosx23+3.893*x23*
         cosx36
A(48,37)=2.060-0.050*x15*x32+4.972*x15*x45-1.742*x29*x44+
         4.039*x42*x19-0.327*x42*x30+2.604*x42*x40+0.827*
         x47*cosx14+1.240*x47*cosx44+3.465*x48*x4*cosx34-
         4.265*x48*x17*cosx2-3.924*x48*x17*cosx4+2.606*
         x48*x17*cosx30+3.196*x48*x17*cosx48+0.267*x48*
         x47*cosx3-3.645*x48*x47*cosx6-3.659*x48*x47*
         cosx15
A(48,38)=2.510+3.562*x47+3.719*x44*x5+0.061*x44*x37-
         2.191*x44*x38+0.611*x39*cosx15+0.962*x39*cosx18-
         3.959*x21*x10*cosx6+1.638*x21*x10*cosx18-1.899*
         x21*x10*cosx34
A(48,39)=4.843-3.400*x1*x47+4.086*x5*cosx32-0.593*x15*
         cosx22+3.842*x15*cosx23-2.589*x15*cosx27
A(48,40)=1.581-2.046*x24+0.559*x31-0.632*x41+0.120*x47-
```

```
             0.774*x1*x24-4.108*x15*x10-0.656*x15*x33
 A(48,41)=4.727+3.157*x41+2.499*x10*x9-1.395*x19*cosx35
 A(48,42)=1.275-2.857*x8-2.282*x12+3.501*x43+0.115*x32*
             x12+1.556*x16*x1*cosx38-4.801*x16*x39*cosx6-
             0.054*x36*x8*cosx9+2.201*x36*x8*cosx21
 A(48,43)=1.428-4.556*x9+3.426*x20*x3+4.538*x20*x27+2.365*
             x26*x17+3.373*x26*x20-2.519*x26*x28-1.939*x23*
             cosx41-2.024*x10*x21*cosx8-4.136*x10*x44*cosx13-
             3.787*x10*x44*cosx20+0.287*x10*x44*cosx29-0.398+
             x10*x44*cosx32-1.693*x10*x44*cosx34
 A(48,44)=1.611-1.920*x21+4.929*x26-2.032*x32+4.083*x6*
             x41-1.360*x12*x2+3.786*x12*x12+2.641*x12*x27+
             4.349*x44*x2+1.237*x45*x5-4.259*x29*cosx20-
             4.991*x29*cosx42-4.887*x29*cosx45-3.154*x40*
             cosx27+4.161*x5*x45*cosx2-4.651*x5*x48*cosx31-
             3.299*x5*x48*cosx35
 A(48,45)=1.603-4.305*x44+2.780*x24*x37+1.277*x39*x6-
             1.906*x39*x44+1.624*x5*cosx1-3.299*x5*cosx12-
             0.242*x5*cosx32+4.847*x5*cosx48
 A(48,46)=0.106+3.290*x1+1.653*x17+2.786*x21-2.949*x47-
             4.764*x45*x18-3.429*x45*x20
 A(48,47)=3.023+0.419*x11-4.570*x3*x12-0.764*x8*x8+0.772*
             x8*x14+3.120*x37*x39-2.171*x5*cosx6+2.083*x5*
             cosx29-3.421*x6*x6*cosx15+4.723*x6*x23*cosx44-
             2.174*x16*x27*cosx38
 A(48,48)=2.519+2.759*x7+1.962*x14+1.477*x3*cosx10+0.650*
             x46*cosx37+3.706*x6*x48*cosx11+2.011*x6*x48*
             cosx28+3.249*x34*x14*cosx13-2.464*x34*x14*
             cosx19+3.755*x34*x14*cosx43+2.520*x34*x16*
             cosx20-0.088*x34*x16*cosx36-2.307*x35*x20*
             cosx23+0.438*x35*x20*cosx35
 b(1)=0.001*x27+0.002*x33+155.092*x40+0.001*x45
 b(2)=0.006*x8+0.003*x14+0.315*u13+0.315*u14
 b(5)=0.009*x10+0.004*x16+0.425*u13+0.425*u14
 b(6)=0.004*x4+0.004*x16+0.002*x22+0.002*x25+0.004*x31+
       0.002*x37+649.630*x44+0.403*u13+0.403*u14
 b(7)=0.658*x20+0.002*x27+0.004*x33+266.985*x40+0.002*x45+
       0.042*u13+0.042*u14
 b(10)=0.005*x2+0.010*x8+0.014*x20+0.003*x23+0.014*x26+
        0.005*x29+132.541*x36+0.009*x38+0.005*x41+0.009*
        x44+0.005*x47+2.404*u13+4.808*u14
 b(11)=59.103*x40
 b(13)=0.003*x30+0.006*x36+0.003*x42
 b(14)=0.002*x23+0.003*x24+0.005*x25+0.005*x29+0.010*x30+
        0.009*x31+117.318*x36+0.005*x37+0.005*x41+0.010*
        x42+1466.134*x44
 b(16)=0.006*x1+37.106*x8+0.006*x13+0.003*x19+0.591*u13+
        0.591*u14
 b(18)=0.004*x10+0.007*x16+0.007*x28+0.004*x34+0.001*x36+
        0.744*u13+0.744*u14
 b(19)=0.003*x3+0.006*x9+94.756*x16+0.006*x21+0.003*x27+
        1.508*u13+1.508*u14
 b(21)=0.002*x27+0.004*x33+0.002*x39+594.201*x46
 b(22)=0.002*x1+0.004*x4+11.699*x8+0.009*x10+0.002*x13+
        0.009*x22+0.004*x28+2.364*u13+2.364*u14
 b(23)=1.459*x4+0.002*x9+0.001*x15+0.117*u13+0.117*u14
 b(24)=0.002*x14+0.003*x20+0.003*x29+0.003*x32+0.007*x35+
        0.002*x38+408.861*x42+0.002*x44+0.003*x47+0.169*
        u13+0.169*u14
 b(25)=0.002*x13+3.114*x20+0.002*x25+0.003*x27+0.006*x33+
        0.003*x39+906.989*x46+0.198*u13+0.198*u14
 b(26)=0.005*x13+0.009*x19+85.506*x26+0.009*x31+0.005*x37+
        0.005*x43+0.454*u13+0.454*u14
 b(27)=0.007*x12+0.004*x18+0.002*x25+0.004*x31+232.502*
        x38+0.002*x43+0.365*u13+0.365*u14
 b(28)=0.002*x28+0.003*x34+0.002*x46
 b(29)=0.001*x26+0.004*x30+0.002*x32+0.008*x36+0.001*x44+
        0.004*x48
```

```
b(30)=0.006*x3+0.003*x5+39.052*x10+0.006*x11+0.006*x15+
      100.685*x18+0.003*x21+0.006*x23+0.003*x29+2.225*
      u13+2.225*u14
b(31)=0.907*x10+0.004*x28+0.002*x30+0.008*x34+0.004*x36+
      0.004*x40+0.002*x48+0.014*u13+0.014*u14
b(33)=0.004*x28+0.007*x34+0.004*x46
b(34)=0.010*x2+0.010*x14+0.005*x20+0.997*u13+0.997*u14
b(35)=0.007*x4+0.003*x14+0.003*x15+0.007*x16+0.006*x20+
      0.007*x21+0.004*x22+65.084*x28+0.006*x32+0.007*x33+
      0.003*x38+0.003*x39+0.003*x44+0.003*x45+1.398*u13+
      1.398*u14
b(37)=0.002*x1+0.003*x2+0.004*x5+13.849*x8+0.009*x11+
      0.002*x12+138.457*x18+0.001*x19+0.007*x20+0.003*
      x26+0.004*x29+0.005*x30+0.002*x36+4.515*u13+6.153*
      u14
b(38)=0.005*x10+0.003*x15+0.010*x16+55.687*x28+0.006*x33+
      0.005*x34+0.003*x39+0.003*x45+1.284*u13+1.284*u14
b(39)=3.637*x4+0.006*x9+0.005*x12+0.003*x15+0.009*x18+
      0.002*x25+0.004*x29+0.009*x30+0.003*x31+0.008*x35+
      0.005*x36+0.002*x37+0.004*x41+505.053*x44+1274.906*
      x48+1.191*u13+1.191*u14
b(40)=0.005*x14+10.902*x16+0.009*x20+0.003*x21+21.298*
      x28+0.002*x32+0.002*x33+0.005*x38+0.001*x39+0.005*
      x44+0.001*x45+0.748*u13+0.748*u14
b(43)=0.005*x6+0.259*x12+0.005*x18+0.003*x24+0.551*u13+
      0.551*u14
b(44)=0.009*x3+0.007*x5+0.007*x6+57.301*x10+46.011*x12+
      0.009*x15+0.007*x17+0.007*x18+0.005*x21+0.004*x23+
      0.003*x24+2.343*u13+2.343*u14
b(45)=0.004*x30+0.009*x36+0.004*x42
b(46)=0.006*x3+40.439*x10+0.008*x15+0.006*x21+29.916*x28+
      0.003*x33+0.002*x39+0.002*x45+0.803*u13+0.803*u14
b(47)=0.003*x7+0.006*x13+9.836*x20+0.006*x25+0.003*x31+
      0.627*u13+0.627*u14
b(48)=3.442*x2+2.240*x4+3.978*x6+7.825*x8+16.098*x10+
      56.124*x12+66.700*x14+71.749*x16+26.547*x18+10.786*
      x20+6.755*x22+1.958*x24+41.158*x26+41.028*x28+
      7.965*x30+39.791*x32+234.509*x34+81.926*x36+
      258.740*x38+608.553*x40+594.032*x42+448.581*x44+
      503.548*x46+949.586*x48+14.976*u13+15.869*u14
```

APPENDIX C

Implementation of the procedures

The following FORTRAN 77 program shows a possible implementation of the generation and solution of benchmark ODE systems. It is used in Chapter 6 for the performance analysis of the algorithm selection procedure. It includes:

1. The main program that defines the physical characteristics of the dynamic structure and calls the appropriate subroutines that are required by each of the 4 tests specified in Figure 6.1.

2. The subroutine BENCHMARK that shows a possible implementation of the procedure that generates the benchmarks shown in the previous section, as presented in Chapter 5.

3. The subroutine AB that shows a possible implementation of the Adams Bashforth algorithm.

4. The subroutine EPSE that shows a possible implementation of the Explicit Power Series Expansion algorithm.

5. Several subroutines from the linear algebra package LINPACK [Dong79] used for the inversion of the matrices A and F.

```
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                     c
c            THIS PROGRAM GENERATES A BENCHMARK ODE SYSTEM AND        c
c                                                                     c
c                 SOLVES IT WITH THE AB OR EPSE ALGORITHM             c
c                                                                     c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
      parameter(inbd=6,in=12)
c
      complex cb(inbd,inbd)
      real cu(inbd,inbd)
      real sr(in),se(in)
      real xo(in),uo(in)
      real alfa(in,in,in),beta(in,in,in,in),gama(in,in,in,in)
      real delta(in,in,in,in,in),xi(in,in,in),eta(in,in)
      real af(in,in),ag(in,in)
      integer zz(in,in,in),df
c
      write(6,14)
   14 format('**************** THE PHYSICAL MODEL ***************')
      nbd=6
      df=2
      write(6,1) nbd,df
    1 format(/,'*** Number of bodies = ',i1,' DOF = ',i1)
      read(5,2) ((cb(i,j),j=1,nbd),i=1,nbd)
    2 format(f7.4,2x,f7.3)
      write(6,3)
    3 format(/,'*** The coupling of the bodies')
      write(5,62) ((cb(i,j),j=1,nbd),i=1,nbd)
   62 format(6(f7.4,'+j',f7.3))
      ib=3
      write(6,6) ib
    6 format(/,'*** The excitation applies on the ',i1,'th body')
      uo(df*(ib-1)+1)=1.
      uo(df*(ib-1)+2)=1.
      read(5,512) ((cu(i,j),j=1,nbd),i=1,nbd)
  512 format(6(f3.1,1x))
      write(6,73)
   73 format(/,'*** The coupling between bodies and inputs ')
      write(5,72) ((cu(i,j),j=1,nbd),i=1,nbd)
   72 format(6(f3.1,1x))
c
      write(6,13)
   13 format(/,'***** THE BENCHMARK ODE SYSTEM A(x)dx/dt=b(x,u) *****')
      n=nbd*df
c
c  Sparsities
c
      srmin=.42
      srmax=.58
      do 99 i=1,n
      sr(i)=srmin+(float(n)-float(i))*(srmax-srmin)/(float(n)-1.)
      se(i)=.90
   99 continue
c
c  Size of linear and nonlinear terms
c
      fl=10.
      fn=5.
      call ben(n,nbd,df,ib,cb,cu,uo,xo,sr,se,fl,fn,alfa,beta,gama,delta
     *,eta,xi,zz,af,ag)
c
c  Algorithm
c  al=1 ---> EPSE
c  al=2 ---> AB
```

```
c
      al=1
      if (al.eq.1) go to 200
c
      na=2
      nb=6
      nc=4
      moa=3
      mob=3
      moc=3
      ha=0.001
      hb=0.01
      hc=0.1
      len=10000
      write(6,100) na,nb,nc,moa,mob,moc,ha,hb,hc,cb(1,1)
  100 format(/,'na=',i2,' nb=',i2,' nc=',i2,' moa=',i2,' mob=',i2,' moc
     *=',i2,' ha=',f6.4,' hb=',f6.4,' hc=',f6.4,' s=',f6.3)
      call ab(n,na,nb,nc,moa,mob,moc,ha,hb,hc,len,xo,uo,alfa,beta,gama,
     *delta,eta,xi,zz,af,ag)
      go to 300
c
  200 na=2
      nb=4
      nc=6
      moa=8
      mob=16
      moc=16
      ha=0.001
      hb=0.01
      hc=0.1
      len=10000
      write(6,100) na,nb,nc,moa,mob,moc,ha,hb,hc,cb(1,1)
  100 format(/,'na=',i2,' nb=',i2,' nc=',i2,' moa=',i2,' mob=',i2,' moc
     *=',i2,' ha=',f6.4,' hb=',f6.4,' hc=',f6.4,' s=',f6.3)
      call epse(n,df,na,nb,nc,moa,mob,moc,ha,hb,hc,len,xo,uo,alfa,beta,
     *gama,delta,eta,xi,zz,af,ag)
c
  300 stop
      end
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                    c
c     This subroutine provides a BENCHMARK ODE system in the form    c
c                              .                                     c
c                          A(x)x=AFx+AGu=b(x,u)                      c
c                                                                    c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
      subroutine ben(n,nbd,df,ib,cb,cu,uo,xo,sr,se,fl,fn,alfa,beta,gama,
     *delta,eta,xi,zz,af,ag)
c
      parameter(inbd=6,in=12)
c
      complex cb(inbd,inbd)
      real cu(inbd,inbd)
      real sr(in),se(in)
      real xo(in),uo(in)
      real alfa(in,in,in),beta(in,in,in,in),gama(in,in,in,in)
      real delta(in,in,in,in,in),xi(in,in,in),eta(in,in)
      real a(in,in)
      real f(in,in),ff(in,in),g(in,in),gu(in)
      real af(in,in),ag(in,in)
      integer zz(in,in,in),df,ip(in)
      real q(in)
c
c  Matrix F
c
      do 1500 i=1,nbd
      do 1501 j=1,nbd
      cr=real(cb(i,j))
      ci=aimag(cb(i,j))
      ii=df*i
      jj=df*j
      f(ii,jj)=cr
      f(ii-1,jj)=ci
      f(ii,jj-1)=-ci
      f(ii-1,jj-1)=cr
 1501 continue
 1500 continue
      do 8000 i=1,n
      do 8001 j=1,n
      ff(i,j)=f(i,j)
 8001 continue
 8000 continue
c
c  Matrix G
c
      ii=0
      do 1511 k=1,nbd
      do 1513 l=1,nbd
      ii=(k-1)*df
      jj=(l-1)*df
      do 1510 i=ii+1,ii+df
      do 1512 j=jj+1,jj+df
      g(i,j)=cu(k,l)
      if (i.eq.j) g(i,j)=1.0
 1512 continue
 1510 continue
 1513 continue
 1511 continue
c
c Equilibrium point
c
      call matmul(g,uo,gu,n,n,1)
      do 513 i=1,n
      xo(i)=-gu(i)
  513 continue
c
c  Inversion of F using LINPACK
c
      call sgeco(ff,n,n,ip,cond,q)
      call sgesl(ff,n,n,ip,xo,0)
c
      write(6,292)
  292 format(/,'*** The nonlinear matrix A(x)')
      mm=1
      iseed=1
c
      do 11 i=1,n
      do 10 j=1,n
c
      iseed=irandom(iseed)
      z=float(iseed)/float(2147483648)
      if(z.le.sr(i)) go to 10
      write(6,900) i,j
  900 format('A(',i2,',',i2,') =',$)
c
      iseed=irandom(iseed)
      z=float(iseed)/float(2147483648)
      iseed=irandom(iseed)
      z=float(iseed)/float(2147483648)
      eta(i,j)=fl*z
      a(i,j)=a(i,j)+eta(i,j)
c
      do 30 kl=1,n
      iseed=irandom(iseed)
      z=float(iseed)/float(2147483648)
      if (z.lt.se(i)) go to 30
      iseed=irandom(iseed)
      z=float(iseed)/float(2147483648)
      t=z
      iseed=irandom(iseed)
      z=float(iseed)/float(2147483648)
      s=fl*z
      if (t.ge.0.5) go to 21
      if (t.lt.0.49) go to 20
      xi(i,j,k1)=s
      zz(i,j,k1)=mm
      a(i,j)=a(i,j)+xi(i,j,k1)*disc(zz(i,j,k1),xo(k1))
      mm=mm+1
      go to 30
 20   alfa(i,j,k1)=-s
      go to 811
 21   alfa(i,j,k1)=s
811   a(i,j)=a(i,j)+alfa(i,j,k1)*xo(k1)
 30   continue
c
      do 40 k1=1,n
      iseed=irandom(iseed)
      z=float(iseed)/float(2147483648)
      if (z.le.se(i)) go to 40
      do 41 l1=1,n
      iseed=irandom(iseed)
      z=float(iseed)/float(2147483648)
      if (z.le.se(i)) go to 41
      iseed=irandom(iseed)
      z=float(iseed)/float(2147483648)
      t=z
      iseed=irandom(iseed)
      z=float(iseed)/float(2147483648)
      s=fn*z
      if (t.ge.0.5) go to 23
      beta(i,j,k1,l1)=-s
      go to 813
 23   beta(i,j,k1,l1)=s
813   a(i,j)=a(i,j)+beta(i,j,k1,l1)*xo(k1)*xo(l1)
 41   continue
 40   continue
c
      do 50 k1=1,n
      iseed=irandom(iseed)
      z=float(iseed)/float(2147483648)
      if (z.le.se(i)) go to 50
      do 51 l1=1,n
      iseed=irandom(iseed)
      z=float(iseed)/float(2147483648)
      if (z.le.se(i)) go to 51
      iseed=irandom(iseed)
      z=float(iseed)/float(2147483648)
      t=z
      iseed=irandom(iseed)
      z=float(iseed)/float(2147483648)
      s=fn*z
      if (t.gt.0.5) go to 25
 24   gama(i,j,k1,l1)=-s
      go to 815
 25   gama(i,j,k1,l1)=s
815   a(i,j)=a(i,j)+gama(i,j,k1,l1)*xo(k1)*cosin(xo(l1))
 51   continue
 50   continue
c
      do 60 k1=1,n
      iseed=irandom(iseed)
      z=float(iseed)/float(2147483648)
      if (z.le.se(i)) go to 60
      do 61 l1=1,n
      iseed=irandom(iseed)
      z=float(iseed)/float(2147483648)
      if (z.le.se(i)) go to 61
      do 62 m1=1,n
      iseed=irandom(iseed)
      z=float(iseed)/float(2147483648)
      if (z.le.se(i)) go to 62
      iseed=irandom(iseed)
      z=float(iseed)/float(2147483648)
      t=z
      iseed=irandom(iseed)
      z=float(iseed)/float(2147483648)
      s=fn*z
      if (t.gt.0.5) go to 27
      delta(i,j,k1,l1,m1)=-s
      go to 817
   27 delta(i,j,k1,l1,m1)=s
  817 a(i,j)=a(i,j)+delta(i,j,k1,l1,m1)*xo(k1)*xo(l1)*cosin(xo(m1))
   62 continue
   61 continue
   60 continue
c
      if (eta(i,j).eq.0.) go to 309
      write(6,208) eta(i,j)
  208 format(f7.3,'+',$)
c
  309 do 300 k1=1,n
      if (alfa(i,j,k1).eq.0.) go to 300
      write(6,201) alfa(i,j,k1),k1
  201 format(f7.3,' * x',i2,'+',$)
  300 continue
c
      do 360 k1=1,n
      if (xi(i,j,k1).eq.0.) go to 360
      write(6,2201) xi(i,j,k1),zz(i,j,k1),k1
 2201 format(f7.3,'disc',i2,'(x',i2,')+',$)
  360 continue
c
      do 301 k1=1,n
      do 302 l1=1,n
      if (beta(i,j,k1,l1).eq.0.) go to 302
      write(6,204) beta(i,j,k1,l1),k1,l1
  204 format(f7.3,' * x',i2,' * x',i2,'+',$)
  302 continue
  301 continue
c
      do 401 k1=1,n
      do 402 l1=1,n
      if (gama(i,j,k1,l1).eq.0.) go to 402
      write(6,205) gama(i,j,k1,l1),k1,l1
  205 format(f7.3,' * x',i2,' * cosx',i2,'+',$)
  402 continue
  401 continue
c
      do 501 k1=1,n
      do 502 l1=1,n
      do 600 m1=1,n
      if (delta(i,j,k1,l1,m1).eq.0.) go to 600
      write(6,206) delta(i,j,k1,l1,m1),k1,l1,m1
  206 format(f7.3,' * x',i2,' * x',i2,' * cosx',i2,'+',$)
  600 continue
  502 continue
  501 continue
c
  210 write(6,1001)
 1001 format(/)
```

```
   10  continue
   11  continue
c
       write(6,1800)
 1800  format(/,'*** The vector b(x,u)')
c
       call matmul(a,f,af,n,n,n)
       call matmul(a,g,ag,n,n,n)
c
       do 1801 i=1,n
       write(6,903) i
  903  format(/,'b(',i2,') = ',$)
       do 951 j=1,n
       write(6,901) af(i,j),j
  901  format(f12.3,' * x',i2,'+',$)
  951  continue
       write(6,2001)
 2001  format(/)
       do 961 j=1,n
       write(6,902) ag(i,j),j
  902  format(f12.3,' * u',i2,'+',$)
  961  continue
       write(6,2601)
 2601  format(/)
 1801  continue
c
       return
       end
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                    c
c                This subroutine provides the solution               c
c                                                                    c
c                of the developed benchmark ODE system               c
c                                                                    c
c                   with the MULTIRATE AB ALGORITHM                  c
c                                                                    c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
      subroutine ab(n,na,nb,nc,moa,mob,moc,ha,hb,hc,len,xo,uo,alfa,beta,
     *gama,delta,eta,xi,zz,af,ag)
c
       parameter(in=12,imo=6,itm=10001)
c
       real alfa(in,in,in),beta(in,in,in,in),gama(in,in,in,in)
       real delta(in,in,in,in,in),xi(in,in,in),eta(in,in)
       integer zz(in,in,in)
       real a(in,in),b(in)
       real af(in,in),ag(in,in)
       real uo(in),xo(in)
       real y(in,itm),phi(in,itm),h(in)
       real s(in,imo),ss(imo,imo)
       real ai(in,in),det(2),work(in),q(in)
       integer ip(in)
       real r(in),slope(in)
       integer ne(in)
c
c  Coefficients of different order formulas
c
       ss(1,1)=1.
       ss(2,1)=3./2.
       ss(2,2)=-1./2.
       ss(3,1)=23./12.
       ss(3,2)=-16./12.
       ss(3,3)=5./12.
       ss(4,1)=55./24.
       ss(4,2)=-59./24.
       ss(4,3)=37./24.
      ss(4,4)=-9./24.
      ss(5,1)=1901./720.
      ss(5,2)=-2774./720.
      ss(5,3)=2616./720.
      ss(5,4)=-1274./720.
      ss(5,5)=251./720.
      ss(6,1)=4277./1440.
      ss(6,2)=-7923./1440.
      ss(6,3)=9982./1440.
      ss(6,4)=-7293./1440.
      ss(6,5)=2877./1440.
      ss(6,6)=-475./1440.
c
c  Grouping of the state variables
c
      read(5,93) (ne(i),i=1,n)
   93 format(i2)
c
c  Step sizes and formulas for each group
c
      do 532 i=1,n
      if (i.gt.na) go to 2
      h(ne(i))=ha
      do 5 nu=1,moa
      s(ne(i),nu)=ss(moa,nu)
    5 continue
      go to 532
    2 if (i.gt.(na+nb)) go to 4
      h(ne(i))=hb
      do 6 nu=1,mob
      s(ne(i),nu)=ss(mob,nu)
    6 continue
      go to 532
    4 h(ne(i))=hc
      do 7 nu=1,moc
      s(ne(i),nu)=ss(moc,nu)
    7 continue
  532 continue
c
      do 7000 i=1,n
c
c  Initial value
c
      y(i,1)=xo(i)
c
c  Input
c
      uo(i)=1.05*uo(i)
 7000 continue
c
c  Specification of the rates
c  im1=ha/hc
c  im2=hb/hc
c
      im1=1
      im2=1
c
      do 733 it=1,len
c
      write(6,310) it
  310 format(/,'***** THE SOLUTION AT ',i4,' *****')
c
c  Number of ODEs and order for each case
c
      ng=na
      mo=moa
      if (mod(it,im1).eq.1) ng=na+nb
      if (mod(it,im1).eq.1) mo=mob
```

```
        if (mod(it,im2).eq.1) ng=na+nb+nc
        if (mod(it,im2).eq.1) mo=moc
c
        do 11 i=1,n
c
c  Previous point of the solution
c
        r(i)=y(i,it)
c
c  Evaluation of A
c
        do 230 j=1,n
        a(i,j)=eta(i,j)
        do 231 k1=1,n
        a(i,j)=a(i,j)+xi(i,j,k1)*disc(zz(i,j,k1),y(k1,it))
        a(i,j)=a(i,j)+alfa(i,j,k1)*y(k1,it)
        do 232 l1=1,n
        a(i,j)=a(i,j)+beta(i,j,k1,l1)*y(k1,it)*y(l1,it)
        a(i,j)=a(i,j)+gama(i,j,k1,l1)*y(k1,it)*cosin(3.14*y(l1,it)/180.)
        do 233 m1=1,n
        a(i,j)=a(i,j)+delta(i,j,k1,l1,m1)*y(k1,it)*y(l1,it)*cosin(3.14*y(
     *m1,it)/180.)
 233    continue
 232    continue
 231    continue
        ai(i,j)=a(i,j)
 230    continue
c
c  Evaluation of b
c
        b(i)=0.
        do 236 j=1,n
        b(i)=b(i)+af(i,j)*y(j,it)+ag(i,j)*uo(j)
 236    continue
c
  11    continue
c
c  Inversion of A using LINPACK
c
        call sgeco(ai,n,n,ip,cond,q)
        if (mod(it,10).eq.0) write(6,17) cond
  17    format(/,'Condition of the matrix A = ',f15.5)
        call sgedi(ai,n,n,ip,det,work,01)
c
c  Evaluation of derivative of x
c
        do 277 i=1,ng
        phi(ne(i),it)=0.
        do 237 j=1,n
        phi(ne(i),it)=phi(ne(i),it)+ai(ne(i),j)*b(j)
 237    continue
 277    continue
c
c  Evaluation of next point of the solution
c
c
c  Case #1
c
        do 9 i=1,na
c
c  Starting points
c
        if (it.lt.moa) go to 2000
        t=0.
        do 476 nu=1,moa
        t=t+s(ne(i),nu)*phi(ne(i),it-nu+1)
 476    continue
        y(ne(i),it+1)=y(ne(i),it)+h(ne(i))*t
```

```
       slope(ne(i))=(y(ne(i),it+1)-r(ne(i)))/h(ne(i))
       y(ne(i),it+1)=r(ne(i))+slope(ne(i))*ha
       go to 9
c
 2000  y(ne(i),it+1)=y(ne(i),it)+h(ne(i))*phi(ne(i),it)
       slope(ne(i))=(y(ne(i),it+1)-r(ne(i)))/h(ne(i))
       y(ne(i),it+1)=r(ne(i))+slope(ne(i))*ha
c
    9  continue
c
c  Case #2
c
       if (nb.eq.0) go to 5000
       do 2009 i=na+1,na+nb
c
c  Interpolation
c
       if (mod(it,m1).eq.1) go to 3002
       y(ne(i),it+1)=r(ne(i))+slope(ne(i))*ha
       go to 2009
c
c  Starting points
c
 3002  if (it.lt.((mob-1)*m1+1)) go to 3000
       t=0.
       do 2476 nu=1,mob
       t=t+s(ne(i),nu)*phi(ne(i),it-nu+1)
 2476  continue
       y(ne(i),it+1)=y(ne(i),it)+h(ne(i))*t
       slope(ne(i))=(y(ne(i),it+1)-r(ne(i)))/h(ne(i))
       y(ne(i),it+1)=r(ne(i))+slope(ne(i))*ha
       go to 2009
c
 3000  y(ne(i),it+1)=y(ne(i),it)+h(ne(i))*phi(ne(i),it)
       slope(ne(i))=(y(ne(i),it+1)-r(ne(i)))/h(ne(i))
       y(ne(i),it+1)=r(ne(i))+slope(ne(i))*ha
 2009  continue
c
c  Case #3
c
 5000  if (nc.eq.0) go to 5001
       do 3009 i=na+nb+1,na+nb+nc
c
c  Interpolation
c
       if (mod(it,m2).eq.1) go to 5002
       y(ne(i),it+1)=r(ne(i))+slope(ne(i))*ha
       go to 3009
c
c  Starting points
c
 5002  if (it.lt.((moc-1)*m2+1)) go to 4000
       t=0.
       do 3476 nu=1,moc
       t=t+s(ne(i),nu)*phi(ne(i),it-nu+1)
 3476  continue
       y(ne(i),it+1)=y(ne(i),it)+h(ne(i))*t
       slope(ne(i))=(y(ne(i),it+1)-r(ne(i)))/h(ne(i))
       y(ne(i),it+1)=r(ne(i))+slope(ne(i))*ha
       go to 3009
c
 4000  y(ne(i),it+1)=y(ne(i),it)+h(ne(i))*phi(ne(i),it)
       slope(ne(i))=(y(ne(i),it+1)-r(ne(i)))/h(ne(i))
       y(ne(i),it+1)=r(ne(i))+slope(ne(i))*ha
 3009  continue
c
 5001  write(6,12) (i,y(i,it+1),xo(i),h(i),slope(i),i=1,n)
   12  format(1x,i2,1x,f16.5,1x,f16.5,1x,f16.10,1x,f16.10)
```

```
c
  733  continue
c
       return
       end
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                    c
c                  This subroutine provides the solution             c
c                                                                    c
c                  of the developed benchmark ODE system             c
c                                                                    c
c                    with the MULTIRATE EPSE ALGORITHM               c
c                                                                    c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
      subroutine epse(n,df,na,nb,nc,moa,mob,moc,ha,hb,hc,len,xo,u,alfa,b
     *eta,gama,delta,eta,xi,zz,af,ag)
c
       parameter(in=12,imo=33)
c
       real alfa(in,in,in),beta(in,in,in,in),gama(in,in,in,in)
       real delta(in,in,in,in,in),xi(in,in,in),eta(in,in)
       integer zz(in,in,in),df
       real a(in,in,imo),b(in,imo)
       real af(in,in),ag(in,in)
       real xo(in),x(in),uo(in),xd(in,imo),z(in,imo),h(in)
       real ai(in,in),det(2),work(in),q(in)
       integer ip(in)
       real r(in),slope(in)
       integer ne(in)
       real t(imo),s(imo),p(imo)
c
c  Grouping the ODEs
c
       read(5,93)  (ne(i),i=1,n)
   93  format(i2)
c
c  Step sizes for each group
c
       do 532 i=1,n
       h(ne(i))=hc
       if  (i.le.(na+nb)) h(ne(i))=hb
       if  (i.le.na) h(ne(i))=ha
  532  continue
c
       do 7000 i=1,n
c
c  Initial value
c
       x(i)=xo(i)
c
c  Input
c
       uo(i)=1.05*uo(i)
 7000  continue
c
c  Specification of the rates
c  im1=ha/hc
c  im2=hb/hc
c
       im1=10
       im2=100
c
       do 733 it=1,len
       if  (mod(it,10).eq.0) write(6,310) it
  310  format(/,'***** THE SOLUTION AT ',i4,' *****')
c
```

```
c  Number of ODEs and orders for each case
c
       ng=na
       mo=moa
       if (mod(it,im1).eq.1) ng=na+nb
       if (mod(it,im1).eq.I) mo=mob
       if (mod(it,im2).eq.1) ng=na+nb+nc
       if (mod(it,im2).eq.1) mo=moc
c
c  Evaluation of A and b
c
       do 9762 nu=1,mo+1
       do 9761 i=1,n
       do 9760 j=1,n
       a(i,j,nu)=0.
 9760  continue
       b(i,nu)=0.
       z(i,nu)=0.
       xd(i,nu)=0.
 9761  continue
 9762  continue
c
       do 11 i=1,n
c
c  Previous point of the solution
c
       r(i)=x(i)
c
       do 230 j=1,n
       a(i,j,1)=eta(i,j)
       do 231 k1=1,n
       a(i,j,1)=a(i,j,1)+xi(i,j,k1)*disc(zz(i,j,k1),x(k1))
       a(i,j,1)=a(i,j,1)+alfa(i,j,k1)*x(k1)
       do 232 l1=1,n
       a(i,j,1)=a(i,j,1)+beta(i,j,k1,l1)*x(k1)*x(l1)
       a(i,j,1)=a(i,j,1)+gama(i,j,k1,l1)*x(k1)*cosin(x(l1))
       do 233 m1=1,n
       a(i,j,1)=a(i,j,1)+delta(i,j,k1,l1,m1)*x(k1)*x(l1)*cosin(x(m1))
  233  continue
  232  continue
  231  continue
       ai(i,j)=a(i,j,1)
       b(i,1)=b(i,1)+af(i,j)*x(j)+ag(i,j)*uo(j)
  230  continue
c
   11  continue
c
c  Inversion of A using LINPACK
c
       call sgeco(ai,n,n,ip,cond,q)
       if (mod(it,10).eq.0) write(6,17) cond
   17  format(/,'--- Condition of the matrix A = ',f15.5)
       call sgedi(ai,n,n,ip,det,work,01)
c
c  Evaluation of 1st derivative of x
c
       do 277 i=1,n
       xd(i,1)=x(i)
c      xd(i,2)=0.
       do 237 j=1,n
       xd(i,2)=xd(i,2)+ai(i,j)*b(j,1)
  237  continue
  277  continue
c
       if (mo.lt.2) go to 9333
       do 3 nu=3,mo+1
c
       do 1 i=1,n
```

```
      if (i.gt.ng) go to 1000
c
      do 2 j=1,n
c
c  Evaluation of derivatives of A
c
      do 300 k1=1,n
      if (alfa(ne(i),j,k1).eq.0) go to 300
      a(ne(i),j,nu-1)=a(ne(i),j,nu-1)+alfa(ne(i),j,k1)*xd(k1,nu-1)
  300 continue
c
      do 301 k1=1,n
      do 302 l1=1,n
      if (beta(ne(i),j,k1,l1).eq.0) go to 302
      w=1.
      v=0.
      do 101 jj=0,nu-2
      v=v+w*xd(k1,nu-2-jj+1)*xd(l1,jj+1)
      w=w*float(nu-2-jj)/float(jj+1)
  101 continue
      a(ne(i),j,nu-1)=a(ne(i),j,nu-1)+beta(ne(i),j,k1,l1)*v
  302 continue
  301 continue
c
      do 401 k1=1,n
      do 402 l1=1,n
      if (gama(ne(i),j,k1,l1).eq.0) go to 402
      w=1.
      v=0.
      wt=1.
      ws=1.
      do 111 jj=0,nu-2
      if (jj.eq.0) t(jj+1)=cosin(x(l1))
      if (jj.eq.0) go to 9994
      t(jj+1)=0.
      do 222 ll=0,jj-1
      if (ll.eq.0) s(ll+1)=sinus(x(l1))
      if (ll.eq.0) go to 9990
      s(ll+1)=0.
      do 333 nn=0,ll-1
      s(ll+1)=s(ll+1)+ws*xd(l1,ll-nn+1)*t(nn+1)
      ws=ws*float((ll-1-nn))/float(nn+1)
  333 continue
 9990 t(jj+1)=t(jj+1)-wt*xd(l1,jj-ll+1)*s(ll+1)
      wt=wt*float((jj-1-ll))/float(ll+1)
  222 continue
 9994 v=v+w*xd(k1,nu-2-jj+1)*t(jj+1)
      w=w*float((nu-2-jj))/float(jj+1)
  111 continue
      a(ne(i),j,nu-1)=a(ne(i),j,nu-1)+gama(ne(i),j,k1,l1)*v
  402 continue
  401 continue
c
      do 501 k1=1,n
      do 502 l1=1,n
      do 600 m1=1,n
      if (delta(ne(i),j,k1,l1,m1).eq.0) go to 600
      w=1.
      v=0.
      wp=1.
      wt=1.
      ws=1.
      do 444 mm=0,nu-2
      do 888 jj=0,mm
      if (jj.eq.0) t(jj+1)=cosin(x(m1))
      if (jj.eq.0) go to 9991
      t(jj+1)=0.
      do 555 ll=0,jj-1
      if (ll.eq.0) s(ll+1)=sinus(x(ml))
      if (ll.eq.0) go to 9992
      s(ll+1)=0.
      do 666 nn=0,ll-1
      s(ll+1)=s(ll+1)+ws*xd(ml,ll-nn+1)*t(nn+1)
      ws=ws*float((ll-1-nn))/float(nn+1)
  666 continue
 9992 t(jj+1)=t(jj+1)-wt*xd(ml,jj-ll+1)*s(ll+1)
      wt=wt*float((jj-1-ll))/float(ll+1)
  555 continue
 9991 p(mm+1)=p(mm+1)+wp*xd(ll,mm-jj+1)*t(jj+1)
      wp=wp*float((mm-jj))/float(jj+1)
  888 continue
      v=v+w*xd(k1,nu-2-mm+1)*p(mm+1)
      w=w*float((nu-2-mm))/float(mm+1)
  444 continue
      a(ne(i),j,nu-1)=a(ne(i),j,nu-1)+delta(ne(i),j,k1,ll,ml)*v
  600 continue
  502 continue
  501 continue
c
c  Evaluation of derivatives of b
c
      b(ne(i),nu-1)=b(ne(i),nu-1)+af(ne(i),j)*xd(j,nu-1)
c
    2 continue
c
c  Evaluation of derivatives of z
c
      v=0.
      w=float(nu)-2.
      do 6 k=1,nu-2
      do 7 j=1,n
      v=v+w*a(ne(i),j,k+1)*xd(j,nu-1-k+1)
    7 continue
      w=w*float((nu-2-k))/float(k+1)
    6 continue
      z(ne(i),nu-1)=b(ne(i),nu-1)-v
      go to 1
c
 1000 z(ne(i),nu-1)=b(ne(i),nu-1)
c
    1 continue
c
c  Evaluation of higher derivatives of x
c
      do 5 i=1,ng
      do 66 j=1,n
      xd(ne(i),nu)=xd(ne(i),nu)+ai(ne(i),j)*z(j,nu-1)
   66 continue
    5 continue
c
    3 continue
c
c  Evaluation of next point of the solution
c
 9333 do 9 i=1,n
      if (i.gt.ng) go to 3000
      x(ne(i))=xd(ne(i),mo+1)
      nu=mo
  781 x(ne(i))=xd(ne(i),nu)+x(ne(i))*h(ne(i))/float(nu)
      slope(ne(i))=(x(ne(i))-r(ne(i)))/h(ne(i))
      nu=nu-1
      if (nu.gt.0) go to 781
c
c  Interpolation
c
 3000 x(ne(i))=r(ne(i))+slope(ne(i))*ha
```

```
c
      if (mod(it,10).eq.0) write(6,329) ne(i),x(ne(i)),xo(ne(i)),h(ne(i
     *))
 329  format(i2,2x,f8.5,2x,f8.5,2x,f8.5)
   9  continue
c
 733  continue
c
      return
      end
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                    
c    This subroutine computes the product c of the matrices a,b      c
c                                                                    c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc c
c
       subroutine matmul(a,b,c,n,m,l)
c
       real a(n,m),b(m,l),c(n,l),ci
c
      do 1 i=1,n
      do 2 j=1,l
      ci=0.
      do 3 k=1,m
      ci=ci+a(i,k)*b(k,j)
   3  continue
      c(i,j)=ci
   2  continue
   1  continue
c
      return
      end
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c     These are the nonlinear functions with discontinuities         c
c                                                                    c
c    that appear in the elements of the nonlinear matrix A(x)        c
c                                                                    c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc c
c
      function disc(l,x)
      if(x.lt.0.) disc=-1.
      if(x.eq.0.) disc=0.
      if(x.gt.0.) disc=1.
      return
      end
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c                        The cos(x) function                         c
c                                                                    c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc c
c
      function cosin(x)
      y=(x/180.)*3.1415927
      z=amod(y,6.2831853)
      if (z.ge.6.2831853) write(6,1)
   1  format('mod overflow')
      cosin=cos(z)
      return
      end
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c                        The sin(x) function                         c
c                                                                    c
                                                                     c
```

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
        function sinus(x)
        y=(x/180.)*3.1415927
        z=amod(y,6.2831853)
        if (z.ge.6.2831853) write(6,1)
    1   format('mod overflow')
        sinus=sin(z)
        return
        end
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                            c
c                  Random number generator                   c
c                                                            c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
        function irandom(ix)
        integer a,p,ix,b15,b16,xhi,xalo,leftlo,fhi,k
        data a/16807/,b15/32768/,b16/65536/,p/2147483647/
        xhi=ix/b16
        xalo=(ix-xhi*b16)*a
        leftlo=xalo/b16
        fhi=xhi*a+leftlo
        k=fhi/b15
        ix=(((xalo-leftlo*b16)-p)+(fhi-k*b15)*b16)+k
        if (ix.lt.0) ix=ix+p
        irandom=ix
        return
        end
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                            c
c                  LINPACK SUBROUTINES                       c
c                                                            c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
      SUBROUTINE SGECO(A,LDA,N,IPVT,RCOND,Z)
      INTEGER LDA,N,IPVT(1)
      REAL A(LDA,1),Z(1)
      REAL RCOND
C
C     SGECO FACTORS A REAL MATRIX BY GAUSSIAN ELIMINATION
C     AND ESTIMATES THE CONDITION OF THE MATRIX.
C
C     IF  RCOND  IS NOT NEEDED, SGEFA IS SLIGHTLY FASTER.
C     TO SOLVE  A*X = B , FOLLOW SGECO BY SGESL.
C     TO COMPUTE  INVERSE(A)*C , FOLLOW SGECO BY SGESL.
C     TO COMPUTE  DETERMINANT(A) , FOLLOW SGECO BY SGEDI.
C     TO COMPUTE  INVERSE(A) , FOLLOW SGECO BY SGEDI.
C
C     ON ENTRY
C
C        A       REAL(LDA, N)
C                THE MATRIX TO BE FACTORED.
C
C        LDA     INTEGER
C                THE LEADING DIMENSION OF THE ARRAY  A .
C
C        N       INTEGER
C                THE ORDER OF THE MATRIX  A .
C
C     ON RETURN
C
C        A       AN UPPER TRIANGULAR MATRIX AND THE MULTIPLIERS
C                WHICH WERE USED TO OBTAIN IT.
C                THE FACTORIZATION CAN BE WRITTEN  A = L*U  WHERE
C                L  IS A PRODUCT OF PERMUTATION AND UNIT LOWER
```

```
C                 TRIANGULAR MATRICES AND  U  IS UPPER TRIANGULAR.
C
C        IPVT     INTEGER(N)
C                 AN INTEGER VECTOR OF PIVOT INDICES.
C
C        RCOND    REAL
C                 AN ESTIMATE OF THE RECIPROCAL CONDITION OF  A .
C                 FOR THE SYSTEM  A*X = B , RELATIVE PERTURBATIONS
C                 IN  A  AND  B  OF SIZE  EPSILON  MAY CAUSE
C                 RELATIVE PERTURBATIONS IN  X  OF SIZE  EPSILON/RCOND .
C                 IF  RCOND  IS SO SMALL THAT THE LOGICAL EXPRESSION
C                            1.0 + RCOND .EQ. 1.0
C                 IS TRUE, THEN  A  MAY BE SINGULAR TO WORKING
C                 PRECISION.  IN PARTICULAR,  RCOND  IS ZERO  IF
C                 EXACT SINGULARITY IS DETECTED OR THE ESTIMATE
C                 UNDERFLOWS.
C
C        Z        REAL(N)
C                 A WORK VECTOR WHOSE CONTENTS ARE USUALLY UNIMPORTANT.
C                 IF  A  IS CLOSE TO A SINGULAR MATRIX, THEN  Z  IS
C                 AN APPROXIMATE NULL VECTOR IN THE SENSE THAT
C                 NORM(A*Z) = RCOND*NORM(A)*NORM(Z) .
C
C     LINPACK. THIS VERSION DATED 08/14/78 .
C     CLEVE MOLER, UNIVERSITY OF NEW MEXICO, ARGONNE NATIONAL LAB.
C
C     SUBROUTINES AND FUNCTIONS
C
C     LINPACK SGEFA
C     BLAS SAXPY,SDOT,SSCAL,SASUM
C     FORTRAN ABS,AMAX1,SIGN
C
C     INTERNAL VARIABLES
C
      REAL SDOT,EK,T,WK,WKM
      REAL ANORM,S,SASUM,SM,YNORM
      INTEGER INFO,J,K,KB,KP1,L
C
C
C     COMPUTE 1-NORM OF A
C
      ANORM = 0.0E0
      DO 10 J = 1, N
         ANORM = AMAX1(ANORM,SASUM(N,A(1,J),1))
   10 CONTINUE
C
C     FACTOR
C
      CALL SGEFA(A,LDA,N,IPVT,INFO)
C
C     RCOND = 1/(NORM(A)*(ESTIMATE OF NORM(INVERSE(A)))) .
C     ESTIMATE = NORM(Z)/NORM(Y) WHERE  A*Z = Y  AND  TRANS(A)*Y = E .
C     TRANS(A)  IS THE TRANSPOSE OF A .  THE COMPONENTS OF  E  ARE
C     CHOSEN TO CAUSE MAXIMUM LOCAL GROWTH IN THE ELEMENTS OF W  WHERE
C     TRANS(U)*W = E .  THE VECTORS ARE FREQUENTLY RESCALED TO AVOID
C     OVERFLOW.
C
C     SOLVE TRANS(U)*W = E
C
      EK = 1.0E0
      DO 20 J = 1, N
         Z(J) = 0.0E0
   20 CONTINUE
      DO 100 K = 1, N
         IF (Z(K) .NE. 0.0E0) EK = SIGN(EK,-Z(K))
         IF (ABS(EK-Z(K)) .LE. ABS(A(K,K))) GO TO 30
            S = ABS(A(K,K))/ABS(EK-Z(K))
            CALL SSCAL(N,S,Z,1)
            EK = S*EK
   30    CONTINUE
         WK = EK - Z(K)
         WKM = -EK - Z(K)
         S = ABS(WK)
         SM = ABS(WKM)
         IF (A(K,K) .EQ. 0.0E0) GO TO 40
            WK = WK/A(K,K)
            WKM = WKM/A(K,K)
         GO TO 50
   40    CONTINUE
            WK = 1.0E0
            WKM = 1.0E0
   50    CONTINUE
         KP1 = K + 1
         IF (KP1 .GT. N) GO TO 90
            DO 60 J = KP1, N
               SM = SM + ABS(Z(J)+WKM*A(K,J))
               Z(J) = Z(J) + WK*A(K,J)
               S = S + ABS(Z(J))
   60       CONTINUE
            IF (S .GE. SM) GO TO 80
               T = WKM - WK
               WK = WKM
               DO 70 J = KP1, N
                  Z(J) = Z(J) + T*A(K,J)
   70          CONTINUE
   80       CONTINUE
   90    CONTINUE
         Z(K) = WK
  100 CONTINUE
      S = 1.0E0/SASUM(N,Z,1)
      CALL SSCAL(N,S,Z,1)
C
C     SOLVE TRANS(L)*Y = W
C
      DO 120 KB = 1, N
         K = N + 1 - KB
         IF (K .LT. N) Z(K) = Z(K) + SDOT(N-K,A(K+1,K),1,Z(K+1),1)
         IF (ABS(Z(K)) .LE. 1.0E0) GO TO 110
            S = 1.0E0/ABS(Z(K))
            CALL SSCAL(N,S,Z,1)
  110    CONTINUE
         L = IPVT(K)
         T = Z(L)
         Z(L) = Z(K)
         Z(K) = T
  120 CONTINUE
      S = 1.0E0/SASUM(N,Z,1)
      CALL SSCAL(N,S,Z,1)
C
      YNORM = 1.0E0
C
C     SOLVE L*V = Y
C
      DO 140 K = 1, N
         L = IPVT(K)
         T = Z(L)
         Z(L) = Z(K)
         Z(K) = T
         IF (K .LT. N) CALL SAXPY(N-K,T,A(K+1,K),1,Z(K+1),1)
         IF (ABS(Z(K)) .LE. 1.0E0) GO TO 130
            S = 1.0E0/ABS(Z(K))
            CALL SSCAL(N,S,Z,1)
            YNORM = S*YNORM
  130    CONTINUE
  140 CONTINUE
      S = 1.0E0/SASUM(N,Z,1)
      CALL SSCAL(N,S,Z,1)
      YNORM = S*YNORM
C
C     SOLVE  U*Z = V
C
      DO 160 KB = 1, N
         K = N + 1 - KB
         IF (ABS(Z(K)) .LE. ABS(A(K,K))) GO TO 150
            S = ABS(A(K,K))/ABS(Z(K))
            CALL SSCAL(N,S,Z,1)
            YNORM = S*YNORM
  150    CONTINUE
         IF (A(K,K) .NE. 0.0E0) Z(K) = Z(K)/A(K,K)
         IF (A(K,K) .EQ. 0.0E0) Z(K) = 1.0E0
         T = -Z(K)
         CALL SAXPY(K-1,T,A(1,K),1,Z(1),1)
  160 CONTINUE
C     MAKE ZNORM = 1.0
      S = 1.0E0/SASUM(N,Z,1)
      CALL SSCAL(N,S,Z,1)
      YNORM = S*YNORM
C
      IF (ANORM .NE. 0.0E0) RCOND = YNORM/ANORM
      IF (ANORM .EQ. 0.0E0) RCOND = 0.0E0
      RETURN
      END
      SUBROUTINE SGEFA(A,LDA,N,IPVT,INFO)
      INTEGER LDA,N,IPVT(1),INFO
      REAL A(LDA,1)
C
C     SGEFA FACTORS A REAL MATRIX BY GAUSSIAN ELIMINATION.
C
C     SGEFA IS USUALLY CALLED BY SGECO, BUT IT CAN BE CALLED
C     DIRECTLY WITH A SAVING IN TIME IF  RCOND  IS NOT NEEDED.
C     (TIME FOR SGECO) = (1 + 9/N)*(TIME FOR SGEFA) .
C
C     ON ENTRY
C
C        A       REAL(LDA, N)
C                THE MATRIX TO BE FACTORED.
C
C        LDA     INTEGER
C                THE LEADING DIMENSION OF THE ARRAY  A .
C
C        N       INTEGER
C                THE ORDER OF THE MATRIX  A .
C
C     ON RETURN
C
C        A       AN UPPER TRIANGULAR MATRIX AND THE MULTIPLIERS
C                WHICH WERE USED TO OBTAIN IT.
C                THE FACTORIZATION CAN BE WRITTEN  A = L*U  WHERE
C                L  IS A PRODUCT OF PERMUTATION AND UNIT LOWER
C                TRIANGULAR MATRICES AND  U  IS UPPER TRIANGULAR.
C
C        IPVT    INTEGER(N)
C                AN INTEGER VECTOR OF PIVOT INDICES.
C
C        INFO    INTEGER
C                = 0  NORMAL VALUE.
C                = K  IF  U(K,K) .EQ. 0.0 .  THIS IS NOT AN ERROR
C                     CONDITION FOR THIS SUBROUTINE, BUT IT DOES
C                     INDICATE THAT SGESL OR SGEDI WILL DIVIDE BY ZERO
C                     IF CALLED.  USE  RCOND  IN SGECO FOR A RELIABLE
C                     INDICATION OF SINGULARITY.
C
C     LINPACK. THIS VERSION DATED 08/14/78 .
C     CLEVE MOLER, UNIVERSITY OF NEW MEXICO, ARGONNE NATIONAL LAB.
```

```
C
C     SUBROUTINES AND FUNCTIONS
C
C     BLAS SAXPY,SSCAL,ISAMAX
C
C     INTERNAL VARIABLES
C
      REAL T
      INTEGER ISAMAX,J,K,KP1,L,NM1
C
C
C     GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING
C
      INFO = 0
      NM1 = N - 1
      IF (NM1 .LT. 1) GO TO 70
      DO 60 K = 1, NM1
         KP1 = K + 1
C
C        FIND L = PIVOT INDEX
C
         L = ISAMAX(N-K+1,A(K,K),1) + K - 1
         IPVT(K) = L
C
C        ZERO PIVOT IMPLIES THIS COLUMN ALREADY TRIANGULARIZED
C
         IF (A(L,K) .EQ. 0.0E0) GO TO 40
C
C           INTERCHANGE IF NECESSARY
C
            IF (L .EQ. K) GO TO 10
               T = A(L,K)
               A(L,K) = A(K,K)
               A(K,K) = T
   10       CONTINUE
C
C           COMPUTE MULTIPLIERS
C
            T = -1.0E0/A(K,K)
            CALL SSCAL(N-K,T,A(K+1,K),1)
C
C           ROW ELIMINATION WITH COLUMN INDEXING
C
            DO 30 J = KP1, N
               T = A(L,J)
               IF (L .EQ. K) GO TO 20
                  A(L,J) = A(K,J)
                  A(K,J) = T
   20          CONTINUE
               CALL SAXPY(N-K,T,A(K+1,K),1,A(K+1,J),1)
   30       CONTINUE
         GO TO 50
   40    CONTINUE
            INFO = K
   50    CONTINUE
   60 CONTINUE
   70 CONTINUE
      IPVT(N) = N
      IF (A(N,N) .EQ. 0.0E0) INFO = N
      RETURN
      END
      SUBROUTINE SGEDI(A,LDA,N,IPVT,DET,WORK,JOB)
      INTEGER LDA,N,IPVT(1),JOB
      REAL A(LDA,1),DET(2),WORK(1)
C
C     SGEDI COMPUTES THE DETERMINANT AND INVERSE OF A MATRIX
C     USING THE FACTORS COMPUTED BY SGECO OR SGEFA.
C
```

```
C     ON ENTRY
C
C        A       REAL(LDA, N)
C                THE OUTPUT FROM SGECO OR SGEFA.
C
C        LDA     INTEGER
C                THE LEADING DIMENSION OF THE ARRAY  A .
C
C        N       INTEGER
C                THE ORDER OF THE MATRIX  A .
C
C        IPVT    INTEGER(N)
C                THE PIVOT VECTOR FROM SGECO OR SGEFA.
C
C        WORK    REAL(N)
C                WORK VECTOR.  CONTENTS DESTROYED.
C
C        JOB     INTEGER
C                = 11   BOTH DETERMINANT AND INVERSE.
C                = 01   INVERSE ONLY.
C                = 10   DETERMINANT ONLY.
C
C     ON RETURN
C
C        A       INVERSE OF ORIGINAL MATRIX IF REQUESTED.
C                OTHERWISE UNCHANGED.
C
C        DET     REAL(2)
C                DETERMINANT OF ORIGINAL MATRIX IF REQUESTED.
C                OTHERWISE NOT REFERENCED.
C                DETERMINANT = DET(1) * 10.0**DET(2)
C                WITH  1.0 .LE. ABS(DET(1)) .LT. 10.0
C                OR  DET(1) .EQ. 0.0 .
C
C     ERROR CONDITION
C
C        A DIVISION BY ZERO WILL OCCUR IF THE INPUT FACTOR CONTAINS
C        A ZERO ON THE DIAGONAL AND THE INVERSE IS REQUESTED.
C        IT WILL NOT OCCUR IF THE SUBROUTINES ARE CALLED CORRECTLY
C        AND IF SGECO HAS SET RCOND .GT. 0.0 OR SGEFA HAS SET
C        INFO .EQ. 0 .
C
C     LINPACK. THIS VERSION DATED 08/14/78 .
C     CLEVE MOLER, UNIVERSITY OF NEW MEXICO, ARGONNE NATIONAL LAB.
C
C     SUBROUTINES AND FUNCTIONS
C
C     BLAS SAXPY,SSCAL,SSWAP
C     FORTRAN ABS,MOD
C
C     INTERNAL VARIABLES
C
      REAL T
      REAL TEN
      INTEGER I,J,K,KB,KP1,L,NM1
C
C
C     COMPUTE DETERMINANT
C
      IF (JOB/10 .EQ. 0) GO TO 70
         DET(1) = 1.0E0
         DET(2) = 0.0E0
         TEN = 10.0E0
         DO 50 I = 1, N
            IF (IPVT(I) .NE. I) DET(1) = -DET(1)
            DET(1) = A(I,I)*DET(1)
C        ...EXIT
            IF (DET(1) .EQ. 0.0E0) GO TO 60
```

```
   10        IF (ABS(DET(1)) .GE. 1.0E0) GO TO 20
                DET(1) = TEN*DET(1)
                DET(2) = DET(2) - 1.0E0
             GO TO 10
   20        CONTINUE
   30        IF (ABS(DET(1)) .LT. TEN) GO TO 40
                DET(1) = DET(1)/TEN
                DET(2) = DET(2) + 1.0E0
             GO TO 30
   40        CONTINUE
   50     CONTINUE
   60     CONTINUE
   70 CONTINUE
C
C     COMPUTE INVERSE(U)
C
      IF (MOD(JOB,10) .EQ. 0) GO TO 150
         DO 100 K = 1, N
            A(K,K) = 1.0E0/A(K,K)
            T = -A(K,K)
            CALL SSCAL(K-1,T,A(1,K),1)
            KP1 = K + 1
            IF (N .LT. KP1) GO TO 90
            DO 80 J = KP1, N
               T = A(K,J)
               A(K,J) = 0.0E0
               CALL SAXPY(K,T,A(1,K),1,A(1,J),1)
   80       CONTINUE
   90       CONTINUE
  100    CONTINUE
C
C        FORM INVERSE(U)*INVERSE(L)
C
         NM1 = N - 1
         IF (NM1 .LT. 1) GO TO 140
         DO 130 KB = 1, NM1
            K = N - KB
            KP1 = K + 1
            DO 110 I = KP1, N
               WORK(I) = A(I,K)
               A(I,K) = 0.0E0
  110       CONTINUE
            DO 120 J = KP1, N
               T = WORK(J)
               CALL SAXPY(N,T,A(1,J),1,A(1,K),1)
  120       CONTINUE
            L = IPVT(K)
            IF (L .NE. K) CALL SSWAP(N,A(1,K),1,A(1,L),1)
  130    CONTINUE
  140    CONTINUE
  150 CONTINUE
      RETURN
      END
c
c
cccc BLAS cccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c
      INTEGER FUNCTION ISAMAX(N,SX,INCX)
C
C     FINDS THE INDEX OF ELEMENT HAVING MAX. ABSOLUTE VALUE.
C     JACK DONGARRA, LINPACK, 3/11/78.
C
      REAL SX(1),SMAX
      INTEGER I,INCX,IX,N
C
      ISAMAX = 0
      IF( N .LT. 1 ) RETURN
      ISAMAX = 1
      IF(N.EQ.1)RETURN
      IF(INCX.EQ.1)GO TO 20
C
C        CODE FOR INCREMENT NOT EQUAL TO 1
C
      IX = 1
      SMAX = ABS(SX(1))
      IX = IX + INCX
      DO 10 I = 2,N
         IF(ABS(SX(IX)).LE.SMAX) GO TO 5
         ISAMAX = I
         SMAX = ABS(SX(IX))
    5    IX = IX + INCX
   10 CONTINUE
      RETURN
C
C        CODE FOR INCREMENT EQUAL TO 1
C
   20 SMAX = ABS(SX(1))
      DO 30 I = 2,N
         IF(ABS(SX(I)).LE.SMAX) GO TO 30
         ISAMAX = I
         SMAX = ABS(SX(I))
   30 CONTINUE
      RETURN
      END
      REAL FUNCTION SASUM(N,SX,INCX)
C
C     TAKES THE SUM OF THE ABSOLUTE VALUES.
C     USES UNROLLED LOOPS FOR INCREMENT EQUAL TO ONE.
C     JACK DONGARRA, LINPACK, 3/11/78.
C
      REAL SX(1),STEMP
      INTEGER I,INCX,M,MP1,N,NINCX
C
      SASUM = 0.0E0
      STEMP = 0.0E0
      IF(N.LE.0)RETURN
      IF(INCX.EQ.1)GO TO 20
C
C        CODE FOR INCREMENT NOT EQUAL TO 1
C
      NINCX = N*INCX
      DO 10 I = 1,NINCX,INCX
        STEMP = STEMP + ABS(SX(I))
   10 CONTINUE
      SASUM = STEMP
      RETURN
C
C        CODE FOR INCREMENT EQUAL TO 1
C
C
C        CLEAN-UP LOOP
C
   20 M = MOD(N,6)
      IF( M .EQ. 0 ) GO TO 40
      DO 30 I = 1,M
        STEMP = STEMP + ABS(SX(I))
   30 CONTINUE
      IF( N .LT. 6 ) GO TO 60
   40 MP1 = M + 1
      DO 50 I = MP1,N,6
        STEMP = STEMP + ABS(SX(I)) + ABS(SX(I + 1)) + ABS(SX(I + 2))
     *  + ABS(SX(I + 3)) + ABS(SX(I + 4)) + ABS(SX(I + 5))
   50 CONTINUE
   60 SASUM = STEMP
      RETURN
      END
      SUBROUTINE SAXPY(N,SA,SX,INCX,SY,INCY)
C
C     CONSTANT TIMES A VECTOR PLUS A VECTOR.
C     USES UNROLLED LOOP FOR INCREMENTS EQUAL TO ONE.
C     JACK DONGARRA, LINPACK, 3/11/78.
C
      REAL SX(1),SY(1),SA
      INTEGER I,INCX,INCY,IX,IY,M,MP1,N
C
      IF(N.LE.0)RETURN
      IF (SA .EQ. 0.0) RETURN
      IF(INCX.EQ.1.AND.INCY.EQ.1)GO TO 20
C
C        CODE FOR UNEQUAL INCREMENTS OR EQUAL INCREMENTS
C          NOT EQUAL TO 1
C
      IX = 1
      IY = 1
      IF(INCX.LT.0)IX = (-N+1)*INCX + 1
      IF(INCY.LT.0)IY = (-N+1)*INCY + 1
      DO 10 I = 1,N
        SY(IY) = SY(IY) + SA*SX(IX)
        IX = IX + INCX
        IY = IY + INCY
   10 CONTINUE
      RETURN
C
C        CODE FOR BOTH INCREMENTS EQUAL TO 1
C
C
C        CLEAN-UP LOOP
C
   20 M = MOD(N,4)
      IF( M .EQ. 0 ) GO TO 40
      DO 30 I = 1,M
        SY(I) = SY(I) + SA*SX(I)
   30 CONTINUE
      IF( N .LT. 4 ) RETURN
   40 MP1 = M + 1
      DO 50 I = MP1,N,4
        SY(I) = SY(I) + SA*SX(I)
        SY(I + 1) = SY(I + 1) + SA*SX(I + 1)
        SY(I + 2) = SY(I + 2) + SA*SX(I + 2)
        SY(I + 3) = SY(I + 3) + SA*SX(I + 3)
   50 CONTINUE
      RETURN
      END
      REAL FUNCTION SDOT(N,SX,INCX,SY,INCY)
C
C     FORMS THE DOT PRODUCT OF TWO VECTORS.
C     USES UNROLLED LOOPS FOR INCREMENTS EQUAL TO ONE.
C     JACK DONGARRA, LINPACK, 3/11/78.
C
      REAL SX(1),SY(1),STEMP
      INTEGER I,INCX,INCY,IX,IY,M,MP1,N
C
      STEMP = 0.0E0
      SDOT = 0.0E0
      IF(N.LE.0)RETURN
      IF(INCX.EQ.1.AND.INCY.EQ.1)GO TO 20
C
C        CODE FOR UNEQUAL INCREMENTS OR EQUAL INCREMENTS
C          NOT EQUAL TO 1
C
      IX = 1
      IY = 1
      IF(INCX.LT.0)IX = (-N+1)*INCX + 1
      IF(INCY.LT.0)IY = (-N+1)*INCY + 1
      DO 10 I = 1,N
        STEMP = STEMP + SX(IX)*SY(IY)
        IX = IX + INCX
        IY = IY + INCY
   10 CONTINUE
      SDOT = STEMP
      RETURN
C
C        CODE FOR BOTH INCREMENTS EQUAL TO 1
C
C
C        CLEAN-UP LOOP
C
   20 M = MOD(N,5)
      IF( M .EQ. 0 ) GO TO 40
      DO 30 I = 1,M
        STEMP = STEMP + SX(I)*SY(I)
   30 CONTINUE
      IF( N .LT. 5 ) GO TO 60
   40 MP1 = M + 1
      DO 50 I = MP1,N,5
        STEMP = STEMP + SX(I)*SY(I) + SX(I + 1)*SY(I + 1) +
     *   SX(I + 2)*SY(I + 2) + SX(I + 3)*SY(I + 3) + SX(I + 4)*SY(I + 4)
   50 CONTINUE
   60 SDOT = STEMP
      RETURN
      END
      SUBROUTINE  SSCAL(N,SA,SX,INCX)
C
C     SCALES A VECTOR BY A CONSTANT.
C     USES UNROLLED LOOPS FOR INCREMENT EQUAL TO 1.
C     JACK DONGARRA, LINPACK, 3/11/78.
C
      REAL SA,SX(1)
      INTEGER I,INCX,M,MP1,N,NINCX
C
      IF(N.LE.0)RETURN
      IF(INCX.EQ.1)GO TO 20
C
C        CODE FOR INCREMENT NOT EQUAL TO 1
C
      NINCX = N*INCX
      DO 10 I = 1,NINCX,INCX
        SX(I) = SA*SX(I)
   10 CONTINUE
      RETURN
C
C        CODE FOR INCREMENT EQUAL TO 1
C
C
C        CLEAN-UP LOOP
C
   20 M = MOD(N,5)
      IF( M .EQ. 0 ) GO TO 40
      DO 30 I = 1,M
        SX(I) = SA*SX(I)
   30 CONTINUE
      IF( N .LT. 5 ) RETURN
   40 MP1 = M + 1
      DO 50 I = MP1,N,5
        SX(I) = SA*SX(I)
        SX(I + 1) = SA*SX(I + 1)
        SX(I + 2) = SA*SX(I + 2)
        SX(I + 3) = SA*SX(I + 3)
        SX(I + 4) = SA*SX(I + 4)
   50 CONTINUE
      RETURN
      END
      SUBROUTINE  SSWAP (N,SX,INCX,SY,INCY)
C
C     INTERCHANGES TWO VECTORS.
C     USES UNROLLED LOOPS FOR INCREMENTS EQUAL TO 1.
C     JACK DONGARRA, LINPACK, 3/11/78.
C
      REAL SX(1),SY(1),STEMP
      INTEGER I,INCX,INCY,IX,IY,M,MP1,N
C
      IF(N.LE.0)RETURN
      IF(INCX.EQ.1.AND.INCY.EQ.1)GO TO 20
C
C       CODE FOR UNEQUAL INCREMENTS OR EQUAL INCREMENTS NOT EQUAL
C         TO 1
C
      IX = 1
      IY = 1
      IF(INCX.LT.0)IX = (-N+1)*INCX + 1
      IF(INCY.LT.0)IY = (-N+1)*INCY + 1
      DO 10 I = 1,N
        STEMP = SX(IX)
        SX(IX) = SY(IY)
        SY(IY) = STEMP
        IX = IX + INCX
        IY = IY + INCY
   10 CONTINUE
      RETURN
C
C       CODE FOR BOTH INCREMENTS EQUAL TO 1
C
C
C       CLEAN-UP LOOP
C
   20 M = MOD(N,3)
      IF( M .EQ. 0 ) GO TO 40
      DO 30 I = 1,M
        STEMP = SX(I)
        SX(I) = SY(I)
        SY(I) = STEMP
   30 CONTINUE
      IF( N .LT. 3 ) RETURN
   40 MP1 = M + 1
      DO 50 I = MP1,N,3
        STEMP = SX(I)
        SX(I) = SY(I)
        SY(I) = STEMP
        STEMP = SX(I + 1)
        SX(I + 1) = SY(I + 1)
        SY(I + 1) = STEMP
        STEMP = SX(I + 2)
        SX(I + 2) = SY(I + 2)
        SY(I + 2) = STEMP
   50 CONTINUE
      RETURN
      END
      SUBROUTINE SGESL(A,LDA,N,IPVT,B,JOB)
C
      INTEGER LDA,N,IPVT(1),JOB
      REAL A(LDA,1),B(1)
      REAL SDOT,T
      INTEGER K,KB,L,NM1
C
      NM1=N-1
      IF (JOB.NE.0) GO TO 50
      IF (NM1.LT.1) GO TO 30
      DO 20 K=1,NM1
      L=IPVT(K)
      T=B(L)
      IF (L.EQ.K) GO TO 10
      B(L)=B(K)
      B(K)=T
   10 CONTINUE
      CALL SAXPY(N-K,T,A(K+1,K),1,B(K+1),1)
   20 CONTINUE
   30 CONTINUE
      DO 40 KB=1,N
      K=N+1-KB
      B(K)=B(K)/A(K,K)
      T=-B(K)
      CALL SAXPY(K-1,T,A(1,K),1,B(1),1)
   40 CONTINUE
      GO TO 100
   50 CONTINUE
      DO 60 K=1,N
      T=SDOT(K-1,A(1,K),1,B(1),1)
      B(K)=(B(K)-T)/A(K,K)
   60 CONTINUE
      IF (NM1.LT.1) GO TO 90
      DO 80 KB=1,NM1
      K=N-KB
      B(K)=B(K)+SDOT(N-K,A(K+1,K),1,B(K+1),1)
      L=IPVT(K)
      IF (L.EQ.K) GO TO 70
      T=B(L)
      B(L)=B(K)
      B(K)=T
   70 CONTINUE
   80 CONTINUE
   90 CONTINUE
  100 CONTINUE
      RETURN
      END
```

The following two FORTRAN 77 programs show a possible implementation of some of the modules of the multiprocessor implementation procedure as presented in Chapter 4. It is used in Chapter 7 for the performance analysis of the procedure. The first program includes:

1. The main program that defines the physical characteristics of the dynamic structure and calls the appropriate subroutines that are required by each of the 11 tests specified by Figure 7.2.

2. The subroutine BENCHMARK that shows a possible implementation of the procedure that generates the benchmarks shown in the previous section, as presented in Chapter 5.

3. The subroutines PARTITIONAB and PARTITIONPS that shows a possible implementation of the partitioning procedure, as presented in Section 4.3, for the AB and the EPSE algorithms respectively. These subroutines include a possible implementation of the FFD algorithm for the selection of the number of processors.

The second program includes:

1. The main program that reads the results of the partitioning and calls the appropriate routines for assigning and sequencing.

2. The subroutine ASSIGN that shows a possible implementation of the assigning procedure, as presented in Section 4.4.

3. The subroutine SEQUENCE that shows a possible implementation of the sequencing procedure, as presented in Section 4.5.

```
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c            THIS PROGRAM PROVIDES THE PARTITIONING FOR THE            c
c                                                                      c
c                  MULTIPROCESSOR IMPLEMENTATION                       c
c                                                                      c
c           OF THE EPSE OR THE AB ALGORITHM FOR THE SOLUTION           c
c                                                                      c
c                 OF A DEVELOPED BENCHMARK ODE SYSTEM                  c
c                                                                      c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc c
c
      parameter(inbd=8,in=48,in1=49,int=6)
c
      integer ta(in,in,int),tb(in,2)
      integer ecm(in,in1,in)
      complex cb(inbd,inbd)
      real cu(inbd,inbd)
      real sr(in),se(in)
      real uo(in)
      integer fl,fu,df
c
      write(6,14)
   14 format(/,'*********** THE PHYSICAL MODEL **********')
      nbd=8
      df=6
      write(6,1) nbd,df
```

```
    1  format('--- Number of bodies = ',i1,' with ',i1,' d.o.f. each')
       read(5,2) ((cb(i,j),j=1,nbd),i=1,nbd)
    2  format(f6.3,2x,f6.2)
       write(6,3)
    3  format('--- The coupling of the bodies',/)
       write(6,4) ((cb(i,j),j=1,nbd),i=1,nbd)
    4  format(8(f7.4,'+j',f6.2,1x))
       ib=5
       write(6,6) ib
    6  format('--- The excitation applies on the ',i1,'th body')
       uo(df*(ib-1)+1)=1.
       uo(df*(ib-1)+2)=1.
       read(5,72) ((cu(i,j),j=1,nbd),i=1,nbd)
   72  format(8(f3.1,1x))
       write(6,73)
   73  format(/,'--- The coupling between bodies and inputs',/)
       write(6,74) ((cu(i,j),j=1,nbd),i=1,nbd)
   74  format(8(f3.1,1x))
c
       write(6,52)
   52  format('********** THE BENCHMARK A(x)dx/dt=AFx+AGu *********')
c
       n=nbd*df
c
c Sparsities of the benchmark
c  l=1 ---> linear distribution
c  l=2 ---> exponential distribution
c  l=3 ---> inverse exponential distribution
c  l=4 ---> linear distribution with long task
c
       l=4
c
       srmin=.92
       srmax=.98
       if (l.eq.4) srmin=.97
       if (l.eq.4) srmax=.98
       do 99 i=1,n
       if (l.eq.1.or.l.eq.4) sr(i)=srmin+(float(n)-float(i))*(srmax-srmi
     *n)/(float(n)-1.)
       if (l.eq.2) sr(i)=srmin+((float(n)-float(i))**2.)*(srmax-srmin)/(
     *(float(n)-1.)**2.)
       if (l.eq.3) sr(i)=srmax-((float(i)-1.)**2.)*(srmax-srmin)/((float
     *(n)-1.)**2.)
       se(i)=.99
   99  continue
       if (l.eq.4) sr(n)=.0
       if (l.eq.4) se(n)=.97
c
       call benchmark(n,nbd,df,ib,cb,cu,uo,sr,se,ta,tb,ecm)
c
c Partitioning procedure
c
       al=1
       fl=1
       fu=5
       if (al.eq.2) go to 47
c
c  Algorithm
c  al=1 ---> EPSE
c  al=2 ---> AB
c
       na=12
       nb=24
       nc=12
       moa=8
       mob=16
       moc=16
       ha=0.001
```

```
      hb=0.01
      hc=0.1
      write(6,56) ha,hb,hc
   56 format(/,'*** PARTITION OF THE EPSE ALGORITHM ***',/,'with step s
     *izes ',f5.3,1x,f5.3,1x,f5.3,/)
c     call partitionps(n,na,nb,nc,moa,mob,moc,fl,fu,ta,tb,ecm)
      go to 48
c
c For al=2 ---> AB
c
   47 na=48
      nb=0
      nc=0
      moa=6
      mob=6
      moc=6
      ha=0.001
      hb=0.001
      hc=0.001
      write(6,46) ha,hb,hc
   46 format(//,'*** PARTITION OF THE AB ALGORITHM ***',/,'with step si
     *zes ',f6.4,1x,f6.4,1x,f6.4,/)
c     call partitionab(n,na,nb,nc,moa,mob,moc,fl,fu,ta,tb,ecm)
c
   48 stop
      end
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                      c
c     This subroutine provides a BENCHMARK ODE system in the form      c
c                                  .                                   c
c                              A(x)x=AFx+AGu=b(x,u)                    c
c                                                                      c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
      subroutine benchmark(n,nbd,df,ib,cb,cu,uo,sr,se,ta,tb,ecm)
c
      parameter(inbd=8,in=48,in1=49,int=6)
c
      integer ta(in,in,int),tb(in,2)
      integer ecm(in,in1,in)
      real gama(in,in),delta(in,in,in),theta(in)
      complex cb(inbd,inbd)
      real cu(inbd,inbd)
      real a(in,in),f(in,in),ff(in,in),g(in,in),gu(in)
      real af(in,in),ag(in,in)
      real alfa(in),beta(in,in)
      real xo(in),uo(in)
      real sr(in),se(in)
      integer zz(in),df
c
      n1=n+1
c
c Matrix F
c
      do 1500 i=1,nbd
      do 1501 j=1,nbd
      cr=real(cb(i,j))
      ci=aimag(cb(i,j))
      do 1502 kk=1,df-1,2
      ii=df*i-kk+1
      jj=df*j-kk+1
      f(ii,jj)=cr
      f(ii-1,jj)=ci
      f(ii,jj-1)=-ci
      f(ii-1,jj-1)=cr
 1502 continue
 1501 continue
 1500  continue
       do 8000 i=1,n
       do 8001 j=1,n
       ff(i,j)=f(i,j)
 8001  continue
 8000  continue
c
c Matrix G
c
       ii=0
       do 1511 k=1,nbd
       do 1513 l=1,nbd
       ii=(k-1)*df
       jj=(l-1)*df
       do 1510 i=ii+1,ii+df
       do 1512 j=jj+1,jj+df
       g(i,j)=cu(k,l)
       if (i.eq.j) g(i,j)=1.0
 1512  continue
 1510  continue
 1513  continue
 1511  continue
c
c  Equilibrium point
c
       call matmul(g,uo,gu,n,n,1)
       do 513 i=1,n
       xo(i)=-gu(i)
  513  continue
       call sgeco(ff,n,n,ip,cond,q)
       call sgesl(ff,n,n,ip,xo,0)
c
       write(6,292)
  292  format(/,'--- The nonlinear matrix A(x)')
       mm=1
       iseed=1
c
       do 11 i=1,n
       do 10 j=1,n
c
       iseed=irandom(iseed)
       z=float(iseed)/2147483648.
       if(z.le.sr(i)) go to 10
       write(6,900) i,j
  900  format('A(',i2,',',i2,') =',$)
c
       iseed=irandom(iseed)
       z=float(iseed)/2147483648.
       iseed=irandom(iseed)
       z=float(iseed)/2147483648.
       eta=5.*z
       a(i,j)=a(i,j)+eta
c
       do 30 k1=1,n
       iseed=irandom(iseed)
       z=float(iseed)/2147483648.
       if (z.lt.se(i)) go to 30
       iseed=irandom(iseed)
       z=float(iseed)/2147483648.
       t=z
       iseed=irandom(iseed)
       z=float(iseed)/2147483648.
       iseed=irandom(iseed)
       z=float(iseed)/2147483648.
       s=5.*z
       if (t.ge.0.5) go to 21
       if (t.lt.0.49) go to 20
       theta(k1)=s
```

```
          zz(k1)=mm
          a(i,j)=a(i,j)+theta(k1)*disc(zz(k1),xo(k1))
          mm=mm+1
          go to 30
   20     alfa(k1)=-s
          go to 811
   21     alfa(k1)=s
  811     a(i,j)=a(i,j)+alfa(k1)*xo(k1)
   30     continue
c
          do 40 k1=1,n
          iseed=irandom(iseed)
          z=float(iseed)/2147483648.
          if (z.le.se(i)) go to 40
          do 41 l1=1,n
          iseed=irandom(iseed)
          z=float(iseed)/2147483648.
          if (z.le.se(i)) go to 41
          iseed=irandom(iseed)
          z=float(iseed)/2147483648.
          t=z
          iseed=irandom(iseed)
          z=float(iseed)/2147483648.
          s=5.*z
          if (t.ge.0.5) go to 23
          beta(k1,l1)=-s
          go to 813
   23     beta(k1,l1)=s
  813     a(i,j)=a(i,j)+beta(k1,l1)*xo(k1)*xo(l1)
   41     continue
   40     continue
c
          do 50 k1=1,n
          iseed=irandom(iseed)
          z=float(iseed)/2147483648.
          if (z.le.se(i)) go to 50
          do 51 l1=1,n
          iseed=irandom(iseed)
          z=float(iseed)/2147483648.
          if (z.le.se(i)) go to 51
          iseed=irandom(iseed)
          z=float(iseed)/2147483648.
          t=z
          iseed=irandom(iseed)
          z=float(iseed)/2147483648.
          s=5.*z
          if (t.gt.0.5) go to 25
   24     gama(k1,l1)=-s
          go to 815
   25     gama(k1,l1)=s
  815     a(i,j)=a(i,j)+gama(k1,l1)*xo(k1)*cosin(xo(l1))
   51     continue
   50     continue
c
          do 60 k1=1,n
          iseed=irandom(iseed)
          z=float(iseed)/2147483648.
          if (z.le.se(i)) go to 60
          do 61 l1=1,n
          iseed=irandom(iseed)
          z=float(iseed)/2147483648.
          if (z.le.se(i)) go to 61
          do 62 m1=1,n
          iseed=irandom(iseed)
          z=float(iseed)/2147483648.
          if (z.le.se(i)) go to 62
          iseed=irandom(iseed)
          z=float(iseed)/2147483648.
```

```
      t=z
      iseed=irandom(iseed)
      z=float(iseed)/2147483648.
      s=5.*z
      if (t.gt.0.5) go to 27
      delta(k1,l1,m1)=-s
      go to 817
 27   delta(k1,l1,m1)=s
817   a(i,j)=a(i,j)+delta(k1,l1,m1)*xo(k1)*xo(l1)*cosin(xo(m1))
 62   continue
 61   continue
 60   continue
c
      write(6,208) eta
208   format(f7.3,'+',$)
      ta(i,j,6)=ta(i,j,6)+1
      eta=0.
c
309   do 300 k1=1,n
      if (alfa(k1).eq.0.) go to 300
      write(6,201) alfa(k1),k1
201   format(f7.3,' * x',i2,'+',$)
      ta(i,j,1)=ta(i,j,1)+1
      ecm(i,j,k1)=1
      alfa(k1)=0.
300   continue
c
      do 360 k1=1,n
      if (theta(k1).eq.0.) go to 360
      write(6,2201) theta(k1),zz(k1),k1
2201  format(f7.3,' * disc',i2,'(x',i2,')+',$)
      ta(i,j,5)=ta(i,j,5)+1
      ecm(i,j,k1)=1
      theta(k1)=0.
      zz(k1)=0
360   continue
c
      do 301 k1=1,n
      do 302 l1=1,n
      if (beta(k1,l1).eq.0.) go to 302
      write(6,204) beta(k1,l1),k1,l1
204   format(f7.3,' * x',i2,' * x',i2,'+',$)
      ta(i,j,2)=ta(i,j,2)+1
      ecm(i,j,k1)=1
      ecm(i,j,l1)=1
      beta(k1,l1)=0.
302   continue
301   continue
c
      do 401 k1=1,n
      do 402 l1=1,n
      if (gama(k1,l1).eq.0.) go to 402
      write(6,205) gama(k1,l1),k1,l1
205   format(f7.3,' * x',i2,' * cosx',i2,'+',$)
      ta(i,j,3)=ta(i,j,3)+1
      ecm(i,j,k1)=1
      ecm(i,j,l1)=1
      gama(k1,l1)=0.
402   continue
401   continue
c
      do 501 k1=1,n
      do 502 l1=1,n
      do 600 m1=1,n
      if (delta(k1,l1,m1).eq.0.) go to 600
      write(6,206) delta(k1,l1,m1),k1,l1,m1
206   format(f7.3,' * x',i2,' * x',i2,' * cosx',i2,'+',$)
      ta(i,j,4)=ta(i,j,4)+1
```

```
          ecm(i,j,k1)=1
          ecm(i,j,l1)=1
          ecm(i,j,m1)=1
          delta(k1,l1,m1)=0.
  600     continue
  502     continue
  501     continue
c
  210     write(6,1001)
 1001     format(/)
   10     continue
   11     continue
c
          write(6,1800)
 1800     format(/,'--- The vector b(x,u)',/)
          call matmul(a,f,af,n,n,n)
          call matmul(a,g,ag,n,n,n)
          do 1801 i=1,n
          write(6,903) i
  903     format(/,'b(',i2,') = ',$)
          do 951 j=1,n
          if (af(i,j).lt.0.001) go to 951
          write(6,901) af(i,j),j
  901     format(f12.3,' * x',i2,'+',$)
          tb(i,1)=tb(i,1)+1
          ecm(i,n1,j)=1
  951     continue
          write(6,2001)
 2001     format(/)
          do 961 j=1,n
          if ((j.lt.13).or.(j.gt.14).or.(ag(i,j).lt.0.001)) go to 961
          write(6,902) ag(i,j),j
  902     format(f12.3,' * u ',i2,'+',$)
          tb(i,2)=tb(i,2)+1
  961     continue
          write(6,2601)
 2601     format(/)
 1801     continue
c
          return
          end
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                   c
c          This subroutine provides the PARTITION into tasks        c
c                                                                   c
c             of the solution of an ODE system A(x)ẋ=b(x,u)         c
c                                                                   c
c                          with the AB algorithm                    c
c                                                                   c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
      subroutine partitionab(n,na,nb,nc,moa,mob,moc,fl,fu,ta,tb,ecm)
c
       parameter(in=48,in1=49,int=6,im=20)
c
       integer ta(in,in,int),tb(in,2)
       integer t(in),tr(in),te(in,in1)
       integer tx(in),tf(in)
       integer ecm(in,in1,in),rcm(in,in)
       integer ne(in)
       integer fl,fu
c
      n1=n+1
      nc1=1
      nc2=0
      nc3=0
      write(6,76) nc1,nc2,nc3
   76  format('nc1,nc2,nc3 =',i2,1x,i2,1x,i2,/)
c
c  Grouping the ODEs
c
       read(5,93)  (ne(i),i=1,n)
   93  format(i2)
c
c Length of tasks
c
       write(6,65)
   65  format(/,'--- Case 1 ---',/)
       if  (nc1.eq.0) go to 900
       do 5 i=1,n
       tr(ne(i))=0
       tx(ne(i))=0
       do 6 j=1,n
       te(ne(i),j)=fl*ta(ne(i),j,1)+2*fl*ta(ne(i),j,2)+(2*fl+fu)*ta(ne(i
     *),j,3)+(3*fl+fu)*ta(ne(i),j,4)+fu*ta(ne(i),j,5)
       tr(ne(i))=tr(ne(i))+te(ne(i),j)
    6  continue
       te(ne(i),n1)=fl*tb(ne(i),1)+fl*tb(ne(i),2)
       tf(ne(i))=n
       if  (i.le.na)  tx(ne(i))=moa+2
       if  (i.gt.na)  tx(ne(i))=tx(ne(i))+2
       tr(ne(i))=tr(ne(i))+te(ne(i),n1)+tf(ne(i))+tx(ne(i))
    5  continue
       do 1002 i=1,n
       t(i)=t(i)+nc1*tr(i)
 1002  continue
  900  write(6,8)
    8  format(/,'--- Length of rows',/)
       write(6,35)  (tr(i),i=1,n)
   35  format(12(i10))
c
       write(6,75)
   75  format(/,'--- Case 2 ---',/)
       if  (nc2.eq.0) go to 901
       do 15 i=1,n
       tr(ne(i))=0
       tx(ne(i))=0
       do 16 j=1,n
       te(ne(i),j)=fl*ta(ne(i),j,1)+2*fl*ta(ne(i),j,2)+(2*fl+fu)*ta(ne(i
     *),j,3)+(3*fl+fu)*ta(ne(i),j,4)+fu*ta(ne(i),j,5)
       tr(ne(i))=tr(ne(i))+te(ne(i),j)
   16  continue
       te(ne(i),n1)=fl*tb(ne(i),1)+fl*tb(ne(i),2)
       tf(ne(i))=n
       if  (i.le.na)  tx(ne(i))=moa+2
       if  (i.le.nb.and.i.gt.na)  tx(ne(i))=mob+2
       if  (i.gt.na)  tx(ne(i))=tx(ne(i))+2
       tr(ne(i))=tr(ne(i))+te(ne(i),n1)+tf(ne(i))+tx(ne(i))
   15  continue
       do 1000 i=1,n
       t(i)=t(i)+nc2*tr(i)
 1000  continue
  901  write(6,18)
   18  format(/,'--- Length of rows',/)
       write(6,45)  (tr(i),i=1,n)
   45  format(12(i10))
c
       write(6,85)
   85  format(/,'--- Case 3 ---',/)
       if  (nc3.eq.0) go to 902
       do 25 i=1,n
       tr(ne(i))=0
       tx(ne(i))=0
       do 26 j=1,n
       te(ne(i),j)=fl*ta(ne(i),j,1)+2*fl*ta(ne(i),j,2)+(2*fl+fu)*ta(ne(i
     *),j,3)+(3*fl+fu)*ta(ne(i),j,4)+fu*ta(ne(i),j,5)
      tr(ne(i))=tr(ne(i))+te(ne(i),j)
   26 continue
      te(ne(i),n1)=fl*tb(ne(i),1)+fl*tb(ne(i),2)
      tf(ne(i))=n
      if (i.le.na) tx(ne(i))=moa+2
      if (i.le.nb.and.i.gt.na) tx(ne(i))=mob+2
      if (i.le.nc.and.i.gt.nb) tx(ne(i))=moc+2
      if (i.gt.na) tx(ne(i))=tx(ne(i))+2
      tr(ne(i))=tr(ne(i))+te(ne(i),n1)+tf(ne(i))+tx(ne(i))
   25 continue
      do 1001 i=1,n
      t(i)=t(i)+nc3*tr(i)
 1001 continue
  902 write(6,28)
   28 format(/,'--- Length of rows',/)
      write(6,55) (tr(i),i=1,n)
   55 format(12(i10))
c
c Coupling between tasks
c
      do 100 i=1,n
      do 200 k=1,n
      do 300 j=1,n1
      if(ecm(i,j,k).eq.1) rcm(i,k)=1
  300 continue
  200 continue
  100 continue
      write(6,117)
  117 format(/,'--- Coupling of tasks',/)
      write(6,416) ((rcm(i,j),i=1,n),j=1,n)
  416 format(48(i1,1x))
c
      write(6,928)
  928 format(/,'--- Total length of rows',/)
      write(6,955) (t(i),i=1,n)
  955 format(12(i12))
c
      return
      end
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                     c
c          This subroutine provides the PARTITION into tasks          c
c                                                                     c
c           of the solution of an ODE system A(x)ẋ=b(x,u)             c
c                                                                     c
c                     with the EPSE algorithm                         c
c                                                                     c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
      subroutine partitionps(n,na,nb,nc,moa,mob,moc,fl,fu,ta,tb,ecm)
c
      parameter(in=48,in1=49,int=6,im=20,imo=16)
c
      integer ta(in,in,int),tb(in,2)
      integer t(in),tr(in),te(in,in1,imo)
      integer tx(in),tz(in,imo),txd(in,imo)
      integer ecm(in,in1,in),rcm(in,in)
      integer ne(in)
      integer fl,fu
c
      n1=n+1
      nc1=90
      nc2=9
      nc3=1
      write(6,76) nc1,nc2,nc3
   76 format('nc1,nc2,nc3 =',i2,1x,i2,1x,i2,/)
```

```
c
c  Grouping the ODEs
c
       read(5,93)  (ne(i),i=1,n)
   93  format(i2)
c
c Length of tasks
c
       write(6,17)
   17  format(/,'--- Case 1 ---',/)
       if (nc1.eq.0) go to 900
       do 5 i=1,n
       tr(ne(i))=0
       do 6 j=1,n
       te(ne(i),j,1)=fl*ta(ne(i),j,1)+2*fl*ta(ne(i),j,2)+(2*fl+fu)*ta(ne
     *(i),j,3)+(3*fl+fu)*ta(ne(i),j,4)+fu*ta(ne(i),j,5)
       tr(ne(i))=tr(ne(i))+te(ne(i),j,1)
    6  continue
       te(ne(i),n1,1)=fl*tb(ne(i),1)+fl*tb(ne(i),2)
       txd(ne(i),1)=n
       tr(ne(i))=tr(ne(i))+te(ne(i),n1,1)+txd(ne(i),1)
       if (i.gt.na) go to 10
       do 2 nu=2,moa
       do 16 j=1,n
       te(ne(i),j,nu)=fl*ta(ne(i),j,1)+2*nu*fl*ta(ne(i),j,2)+(6*nu-6)*fl
     **ta(ne(i),j,3)+(8*nu-6)*fl*ta(ne(i),j,4)
       tr(ne(i))=tr(ne(i))+te(ne(i),j,nu)
   16  continue
       te(ne(i),n1,nu)=fl*tb(ne(i),1)
       if (nu.eq.2) tz(ne(i),nu)=1
       if (nu.gt.2) tz(ne(i),nu)=(nu-1)*(n+1)+1
       txd(ne(i),nu)=n
       tr(ne(i))=tr(ne(i))+te(ne(i),n1,nu)+tz(ne(i),nu)+txd(ne(i),nu)
    2  continue
       tx(ne(i))=3*moa
       tr(ne(i))=tr(ne(i))+tx(ne(i))
       go to 5
   10  do 12 nu=2,moa
       tz(ne(i),nu)=fl*tb(ne(i),1)
       txd(ne(i),nu)=2
       tr(ne(i))=tr(ne(i))+tz(ne(i),nu)+txd(ne(i),nu)
   12  continue
       tx(ne(i))=2
       tr(ne(i))=tr(ne(i))+tx(ne(i))
    5  continue
       do 1002 i=1,n
       t(i)=t(i)+nc1*tr(i)
 1002  continue
  900  write(6,8)
    8  format(/,'--- Length of rows',/)
       write(6,255)  (tr(i),i=1,n)
  255  format(12(i10))
c
       write(6,27)
   27  format(/,'--- Case 2 ---',/)
       if (nc2.eq.0) go to 901
       do 15 i=1,n
       tr(ne(i))=0
       do 46 j=1,n
       te(ne(i),j,1)=fl*ta(ne(i),j,1)+2*fl*ta(ne(i),j,2)+(2*fl+fu)*ta(ne
     *(i),j,3)+(3*fl+fu)*ta(ne(i),j,4)+fu*ta(ne(i),j,5)
       tr(ne(i))=tr(ne(i))+te(ne(i),j,1)
   46  continue
       te(ne(i),n1,1)=fl*tb(ne(i),1)+fl*tb(ne(i),2)
       txd(ne(i),1)=n
       tr(ne(i))=tr(ne(i))+te(ne(i),n1,1)+txd(ne(i),1)
       if (i.gt.(na+nb)) go to 110
       if (i.gt.na) go to 310
```

```
      do 32 nu=2,moa
      do 236 j=1,n
      te(ne(i),j,nu)=fl*ta(ne(i),j,1)+2*nu*fl*ta(ne(i),j,2)+(6*nu-6)*fl
     **ta(ne(i),j,3)+(8*nu-6)*fl*ta(ne(i),j,4)
      tr(ne(i))=tr(ne(i))+te(ne(i),j,nu)
 236  continue
      te(ne(i),n1,nu)=fl*tb(ne(i),1)
      if (nu.eq.2) tz(ne(i),nu)=1
      if (nu.gt.2) tz(ne(i),nu)=(nu-1)*(n+1)+1
      txd(ne(i),nu)=n
      tr(ne(i))=tr(ne(i))+te(ne(i),n1,nu)+tz(ne(i),nu)+txd(ne(i),nu)
  32  continue
      tx(ne(i))=3*moa
      tr(ne(i))=tr(ne(i))+tx(ne(i))
      if (moa.ge.mob) go to 15
      do 42 nu=moa+1,mob
      tz(ne(i),nu)=fl*tb(ne(i),1)
      txd(ne(i),nu)=2
      tr(ne(i))=tr(ne(i))+tz(ne(i),nu)+txd(ne(i),nu)
  42  continue
      go to 15
 310  do 412 nu=2,mob
      do 116 j=1,n
      te(ne(i),j,nu)=fl*ta(ne(i),j,1)+2*nu*fl*ta(ne(i),j,2)+(6*nu-6)*fl
     **ta(ne(i),j,3)+(8*nu-6)*fl*ta(ne(i),j,4)
      tr(ne(i))=tr(ne(i))+te(ne(i),j,nu)
 116  continue
      te(ne(i),n1,nu)=fl*tb(ne(i),1)
      if (nu.eq.2) tz(ne(i),nu)=1
      if (nu.gt.2) tz(ne(i),nu)=(nu-1)*(n+1)+1
      txd(ne(i),nu)=n
      tr(ne(i))=tr(ne(i))+te(ne(i),n1,nu)+tz(ne(i),nu)+txd(ne(i),nu)
 412  continue
      tx(ne(i))=3*mob+2
      tr(ne(i))=tr(ne(i))+tx(ne(i))
      go to 15
 110  do 112 nu=2,mob
      tz(ne(i),nu)=fl*tb(ne(i),1)
      txd(ne(i),nu)=2
      tr(ne(i))=tr(ne(i))+tz(ne(i),nu)+txd(ne(i),nu)
 112  continue
      tx(ne(i))=2
      tr(ne(i))=tr(ne(i))+tx(ne(i))
  15  continue
      do 1000 i=1,n
      t(i)=t(i)+nc2*tr(i)
1000  continue
 901  write(6,18)
  18  format(/,'--- Length of rows',/)
      write(6,155) (tr(i),i=1,n)
 155  format(12(i10))
c
      write(6,37)
  37  format(/,'--- Case 3',/)
      if (nc3.eq.0) go to 902
      do 25 i=1,n
      tr(ne(i))=0
      do 56 j=1,n
      te(ne(i),j,1)=fl*ta(ne(i),j,1)+2*fl*ta(ne(i),j,2)+(2*fl+fu)*ta(ne
     *(i),j,3)+(3*fl+fu)*ta(ne(i),j,4)+fu*ta(ne(i),j,5)
      tr(ne(i))=tr(ne(i))+te(ne(i),j,1)
  56  continue
      te(ne(i),n1,1)=fl*tb(ne(i),1)+fl*tb(ne(i),2)
      txd(ne(i),1)=n
      tr(ne(i))=tr(ne(i))+te(ne(i),n1,1)+txd(ne(i),1)
      if (i.gt.(na+nb)) go to 410
      if (i.gt.na) go to 510
      do 62 nu=2,moa
```

```
        do 66 j=1,n
        te(ne(i),j,nu)=fl*ta(ne(i),j,1)+2*nu*fl*ta(ne(i),j,2)+(6*nu-6)*fl
     **ta(ne(i),j,3)+(8*nu-6)*fl*ta(ne(i),j,4)
        tr(ne(i))=tr(ne(i))+te(ne(i),j,nu)
   66   continue
        te(ne(i),n1,nu)=fl*tb(ne(i),1)
        if (nu.eq.2) tz(ne(i),nu)=1
        if (nu.gt.2) tz(ne(i),nu)=(nu-1)*(n+1)+1
        txd(ne(i),nu)=n
        tr(ne(i))=tr(ne(i))+te(ne(i),n1,nu)+tz(ne(i),nu)+txd(ne(i),nu)
   62   continue
        tx(ne(i))=3*moa
        tr(ne(i))=tr(ne(i))+tx(ne(i))
        if (moa.ge.moc) go to 25
        do 72 nu=moa+1,moc
        tz(ne(i),nu)=fl*tb(ne(i),2)
        txd(ne(i),nu)=2
        tr(ne(i))=tr(ne(i))+tz(ne(i),nu)+txd(ne(i),nu)
   72   continue
        go to 25
  510   do 82 nu=2,mob
        do 86 j=1,n
        te(ne(i),j,nu)=fl*ta(ne(i),j,1)+2*nu*fl*ta(ne(i),j,2)+(6*nu-6)*fl
     **ta(ne(i),j,3)+(8*nu-6)*fl*ta(ne(i),j,4)
        tr(ne(i))=tr(ne(i))+te(ne(i),j,nu)
   86   continue
        te(ne(i),n1,nu)=fl*tb(ne(i),1)
        if (nu.eq.2) tz(ne(i),nu)=1
        if (nu.gt.2) tz(ne(i),nu)=(nu-1)*(n+1)+1
        txd(ne(i),nu)=n
        tr(ne(i))=tr(ne(i))+te(ne(i),n1,nu)+tz(ne(i),nu)+txd(ne(i),nu)
   82   continue
        tx(ne(i))=3*mob+2
        tr(ne(i))=tr(ne(i))+tx(ne(i))
        if (mob.ge.moc) go to 25
        do 92 nu=mob+1,moc
        tz(ne(i),nu)=fl*tb(ne(i),2)
        txd(ne(i),nu)=2
        tr(ne(i))=tr(ne(i))+tz(ne(i),nu)+txd(ne(i),nu)
   92   continue
        go to 25
  410   do 22 nu=2,moc
        do 216 j=1,n
        te(ne(i),j,nu)=fl*ta(ne(i),j,1)+2*nu*fl*ta(ne(i),j,2)+(6*nu-6)*fl
     **ta(ne(i),j,3)+(8*nu-6)*fl*ta(ne(i),j,4)
        tr(ne(i))=tr(ne(i))+te(ne(i),j,nu)
  216   continue
        te(ne(i),n1,nu)=fl*tb(ne(i),1)
        if (nu.eq.2) tz(ne(i),nu)=1
        if (nu.gt.2) tz(ne(i),nu)=(nu-1)*(n+1)+1
        txd(ne(i),nu)=n
        tr(ne(i))=tr(ne(i))+te(ne(i),n1,nu)+tz(ne(i),nu)+txd(ne(i),nu)
   22   continue
        tx(ne(i))=3*moc+2
        tr(ne(i))=tr(ne(i))+tx(ne(i))
   25   continue
        do 1001 i=1,n
        t(i)=t(i)+nc3*tr(i)
 1001   continue
  902   write(6,28)
   28   format(/,'--- Length of rows',/)
        write(6,55) (tr(i),i=1,n)
   55   format(12(i10))
c
c Coupling between tasks
c
        do 100 i=1,n
        do 200 k=1,n
```

```
      do 300 j=1,n1
      if(ecm(i,j,k).eq.1) rcm(i,k)=1
  300 continue
  200 continue
  100 continue
      write(6,57)
   57 format(/,'--- Coupling of tasks',/)
      write(6,416) ((rcm(i,j),i=1,n),j=1,n)
  416 format(48(i1,1x))
c
      write(6,928)
  928 format(/,'--- Total length of rows',/)
      write(6,955) (t(i),i=1,n)
  955 format(12(i12))
c
      return
      end

c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                            c
c       THIS PROGRAM PROVIDES THE ASSIGNMENT AND SEQUENCE     c
c                                                            c
c             FOR THE MULTIPROCESSOR IMPLEMENTATION          c
c                                                            c
c                OF THE AB OR THE EPSE ALGORITHM             c
c                                                            c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
      parameter(in=48,im=48)
c
      integer as(im,in),p(im,in),pr(im),t(in),st(in),f(im)
      integer ntp(im),rcm(in,in)
      integer bst,al,z
c
c Algorithm
c al=1 ---> EPSE
c al=2 ---> AB
c
      al=1
c
c bound on the solution time
c
      if (al.eq.1) bst=400000
      if (al.eq.2) bst=4000
c
c number of solution cases
c
      if (al.eq.1) nc=100
      if (al.eq.2) nc=1
      n=48
c
c number of nonzero elements
c
      ns=5
c
c read dependencies
c
      read(5,2) ((rcm(i,j),i=1,n),j=1,n)
    2 format(48(i1,1x))
c
c FFD algorithm on total tasks
c
      read(5,1) (t(i),i=1,n)
    1 format(12(i12))
      m=1
      do 60 i=1,n
      st(i)=t(i)
   60  continue
       do 22 i=1,n-1
       max=i
       do 23 k=i+1,n
       if (st(max).lt.st(k)) max=k
   23  continue
       it=st(max)
       st(max)=st(i)
       st(i)=it
   22  continue
       z=bst-(nc*(ns**3))/(3*m)
       do 1522 i=1,n
       j=1
 1534  if ((z-pr(j)).ge.st(i)) go to 1533
       j=j+1
       if (j.gt.m) m=m+1
       z=bst-(nc*(ns**3))/(3*m)
       go to 1534
 1533  pr(j)=pr(j)+st(i)
 1522  continue
       write(6,555) m
  555  format(/,'*** Number of processors = ',i2)
c
c Schedule the execution of total tasks
c as=assignment matrix
c ntp=number of tasks per processor
c p=processors with their assigned tasks
c
       read(5,991) (t(i),i=1,n)
  991  format(12(i12))
       if (m.eq.1) go to 90
       call assign(n,m,t,as,ntp,p)
       call sequence(n,m,as,rcm,ntp,p)
       go to 91
   90  ntp(1)=n
       do 92 i=1,n
       as(1,i)=1
       p(1,i)=i
   92  continue
c
c Schedule the excecution for each solution case
c
   91  ll=3
c
   70  write(6,19) ll
   19  format('Case # ',i1,/)
c
c read tasks of case 3,2,1
c
       read(5,81) (t(i),i=1,n)
   81  format(12(i10))
c
c completion times of processors
c
       do 3 j=1,m
       f(j)=(ns**3)/(3*m)
       do 4 i=1,ntp(j)
       f(j)=f(j)+t(p(j,i))
    4  continue
    3  continue
c
c print schedule of excecution
c
      do 21 j=1,m
      write(6,25) j
   25 format('p',i2,'=',$)
      do 32 i=1,ntp(j)
      write(6,10) p(j,i)
   10 format(i2,'+',$)
   32 continue
      write(6,13) f(j)
   13 format(' ---> f=',i10)
   21 continue
      write(6,15)
   15 format(/)
c
c go to next solution case
c
      ll=ll-1
      if (ll.eq.0) go to 71
      go to 70
c
   71 stop
      end
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                    c
c          This subroutine is the implementation of the              c
c                                                                    c
c                          LPT algorithm                             c
c                                                                    c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
      subroutine assign(n,m,t,as,ntp,p)
c
       parameter(in=48,im=48)
c
       integer as(im,in),p(im,in),t(in),st(in),ist(in),ntp(im),f(im)
c
c initialization
c
      do 30 j=1,m
      f(j)=0
      ntp(j)=0
      do 31 i=1,n
      as(j,i)=0
      p(j,i)=0
   31 continue
   30 continue
c
c Sorting of the tasks
c
       do 5 i=1,n
       st(i)=t(i)
    5  continue
       do 61 i=1,n
       max=1
       do 50 l=1,n
       if (st(l).gt.st(max)) max=l
   50  continue
       ist(i)=max
       st(max)=0
   61  continue
c
       do 60 i=1,n
       st(i)=t(i)
   60  continue
       do 22 i=1,n-1
       max=i
       do 23 k=i+1,n
       if (st(max).lt.st(k)) max=k
   23  continue
       it=st(max)
       st(max)=st(i)
       st(i)=it
   22  continue
```

```
c
c assignment
c
      do 3 i=1,n
      min=1
      if (m.eq.1) go to 93
      do 12 j=2,m
      if (f(min).gt.f(j)) min=j
   12 continue
   93 as(min,ist(i))=1
      f(min)=f(min)+st(i)
    3 continue
c
      do 14 j=1,m
      k=1
      do 35 i=1,n
      if (as(j,i).eq.0) go to 35
      p(j,k)=i
      k=k+1
      ntp(j)=ntp(j)+1
   35 continue
   14 continue
c
      return
      end
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                               c
c     This subroutine SEQUENCES the execution of the tasks      c
c                                                               c
c                  assigned to any processor                    c
c                                                               c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
      subroutine sequence(n,m,as,rcm,ntp,p)
c
       parameter(in=48,im=48)
c
      integer dr(im,in),rcm(in,in)
      integer as(im,in),p(im,in)
      integer ntp(im)
c
c Initialization
c
      do 1 j=1,m
      do 2 i=1,n
      p(j,i)=0
    2 continue
    1 continue
c
c Priorities of the tasks
c
      do 5 j=1,m
      do 4 i=1,n
      dr(j,i)=-48
      if (as(j,i).eq.0) go to 4
      dr(j,i)=0
      do 6 k=1,n
      dr(j,i)=dr(j,i)+rcm(i,k)*(1-as(j,k))
    6 continue
    4 continue
    5 continue
c
c Sequence the execution of the tasks by sorting their priorities
c
      do 11 j=1,m
      do 3 i=1,ntp(j)
      l=1
      do 12 k=2,n
      if (dr(j,l).lt.dr(j,k)) l=k
   12 continue
      dr(j,l)=-dr(j,l)
      p(j,i)=l
    3 continue
   11 continue
c
      return
      end
```