# Multiprocessor implementation of algorithms for ordinary differential equations

**Socrates Dimitriadis**
Senior Member of the Technical Staff
*Teledyne Controls Inc.*
12333 West Olympic Blvd.
Los Angeles, CA 90064

**Walter J. Karplus**
Professor
Computer Science Department
*University of Califomia*
Los Angeles, CA 90024

*Multiprocessor computers constitute a cost-effective alternative to supercomputers for applications that require the solution of large sets of ordinary differential equations. The efficient use of multiprocessor computers however, requires tools for generation of parallel applications. This paper presents a general methodology for parallelization of the integration algorithms through a heuristic procedure for static multiprocessor scheduling of the computations involved. The procedure partitions, distributes and schedules the entire computation, based on decomposition of the mathematical models. Both problems are addressed: (i) minimize the completion time for a specified number of processors, and (ii) minimize the number of processors required to achieve a specified performance. The procedure combines high performance and simplicity for a wide variety of mathematical models, numerical algorithms and multiprocessor architectures.*

**Key words:** load balancing, loosely-coupled computation, multiprocessing, ordinary differential equation, scheduling

## Introduction

Along with signal processing and the solution of partial differential equations, the treatment of systems of ordinary differential equations (ODEs) constitutes one of the major applications of scientific computers. Systems of ODEs characterize phenomena in a wide variety of scientific and engineering areas. In many instances, these systems of ODEs may be so large and complex as to pose substantial computational challenges. For example, the mathematical models characterizing complex flexible dynamic structures such as space shuttles, space stations, helicopters and advanced robots may contain hundreds of simultaneous ODEs with many nonlinear and time-varying parameters. This is also frequently the case in parameter identification, in the optimization of control systems and in the transient analysis of VLSI circuits. Fluid and thermal dynamic problems, whose models are described by partial differential equations, often require the solution of thousands of simultaneous ODEs, when solved by collocation methods or by the method of lines.

The solution of such ODE systems, particularly in real-time, may tax the capabilities of even the largest main-frame computers and supercomputers. Multiprocessor computers constitute a cost-effective alternative to such expensive computer systems, and since 1985 over a dozen different multiprocessor computers have appeared on the market (Karplus 1987). Multiprocessor architectures appear to be particularly suitable for the implementation of simulation models characterized by ODEs, since such models represent system components functioning concurrently, *i.e.* in parallel. The effective use of multiprocessor computers as simulators requires software tools to generate parallel application software. A major problem in parallelizing the applications is the scheduling of the computation on the processing elements in a manner that effectively exploits the parallelism inherent

in the model and system of ODEs. Most contemporary multiprocessor computers are accompanied by compilers which translate the sequential programs that describe the portions of the application to be executed on each of the processors into machine code. However, no decision is incorporated on how the application should be partitioned. Some multiprocessor computers have compilers which parallelize sequential programs that describe the entire application to produce machine code adapted to the characteristics of the multiprocessor architecture. The scheduler described in this paper on the other hand, does not employ a sequential program as its input. Rather it partitions the system of ODEs into subsystems and each processor is assigned one of these subsystems. Only after the scheduling operation has taken place and the programs to be executed concurrently have been generated is a compiler employed to translate the programs into machine code executable by each processor.

In most scientific and engineering applications involving ODEs it is desired primarily to explore the effect on the model outputs of changes in initial conditions, parameter values, and external excitations. Only very rarely is it required to make substantial changes in the mathematic model. It is feasible therefore to employ a static scheduling method. In such an approach, a suitable resource allocation schedule is determined off-line and well in advance of the solution runs. The software system to solve the scheduling problem may be implemented on a conventional computer and may, if necessary, require a relatively long time to arrive at an acceptable scheduling scheme.

Scheduling problems are NP-complete and it is therefore necessary to accept heuristic, sub-optimal solutions. General purpose static schedulers are likely to be very inefficient for large systems of ODEs. There is a need for a specialized scheduler that exploits the specific characteristics of such applications. It is the purpose of this paper to present a general heuristic methodology for the realization of a static scheduler specialized for computations involving the solution of large sets of ODEs. The approach is based upon the decomposition of the mathematical models, a technique termed *equation segmentation* (Franklin 1978).

The numerical solution of ODEs requires the discretization of the domain of the independent variable t, with a discretization interval called *step size*, and the generation of solutions only for the corresponding discretization points. The time required by the computer to move the solution one step ahead (i.e. to compute the next solution point) is called *completion time*. With the exception of a few special cases, each point of the solution requires the same computation. It is therefore, sufficient to optimize the schedule of the computation for one point of the solution only.

There are a number of motivations for scheduling ODE applications. The scheduling procedure presented in this paper is designed to determine the minimum number of processors that are required to attain a certain speed of the solution of the ODE system, i.e. to achieve a completion time that meets a specified *deadline*. If desired, the minimum completion time for a specified number of processors can also be determined by the scheduler. The scheduling procedure partitions the entire computation into a number of tasks and determines the minimum number of processors to attain a specified performance. It then distributes the tasks among the processors and sequences the execution of the tasks on each processor to minimize the completion time. It is designed to combine high performance with simplicity for a wide variety of models, numerical algorithms, and multiprocessor architectures.

## Mathematical Models and Integration Algorithms

Most models of ODE applications take the explicit state equations form:

$$\dot{x} = f(x,t) \tag{1}$$

Some models take the semi-explicit form:

$$A(x,t)\dot{x} = b(x,t) \tag{2}$$

Less frequently models take the general implicit form:

$$\Phi(\dot{x},x,t) = 0 \tag{3}$$

Note that (2) can be considered as a special case of (3), and (1) can be considered to be a special case of (2).

In most high-speed applications, it is impractical to vary the step size in the course of a solution run. However, it is often advantageous to employ multiple step sizes. In such *multi-rate* algorithms, the ODEs to be integrated are divided into groups, and a different step size is chosen for each group. Then the computation differs from time level to time level, since at certain time levels all equations are integrated while at others only the ones with shorter step sizes are fully processed. Finally, where there exist discontinuities in the model, a more elaborate procedure is required in order to locate the switching points (points of discontinuity) before evaluating the new point of the solution.

A wide variety of algorithms are available for the integration of ODEs. The most widely used of these are listed in Table 1. All three forms of models are treated in the same way by any numerical algorithm, with the difference being the computation required in evaluating the derivatives of the dependent variables x.

**Table 1.** Algorithms for the Integration of Ordinary Differential Equations.

| ALGORITHM | EXAMPLES |
|---|---|
| Explicit One-step Single-derivative | Euler<br>Explicit Runge Kutta<br>Runge Kutta Fehlberg |
| Explicit Multi-step Single-derivative | Adams Bashforth |
| Explicit One-step Multi-derivative | Explicit Power Series Expansion |
| Implicit One-step Single-derivative | Implicit Runge Kutta<br>Trapezoidal |
| Implicit Multi-step Single-derivative | Adams Moulton<br>Adams Moulton Predictor Corrector<br>Gear |
| Implicit One-step Multi-derivative | Implicit Power Series Expansion |
| Implicit Multi-step Multi-derivative | Enright |
| Extrapolation | Richardson |

The computations at one time level in the solution of the set of ODEs can be conveniently represented in the form of a precedence graph to show the sequentialities inherent in the computation. Despite the differences of the algorithms their computational graphs have a similar structure. Figure 1 shows such a graph for a set of ODEs in the form of (2), when integrated using a single-rate Explicit Power Series Expansion (EPSE) algorithm (Halin 1983). Note that the graph has the pattern of an array with N columns (one for each of the M dependent variables x) and rows (one for each of the m derivatives of the dependent variables). If different orders are used for different ODEs, the lengths of the columns are

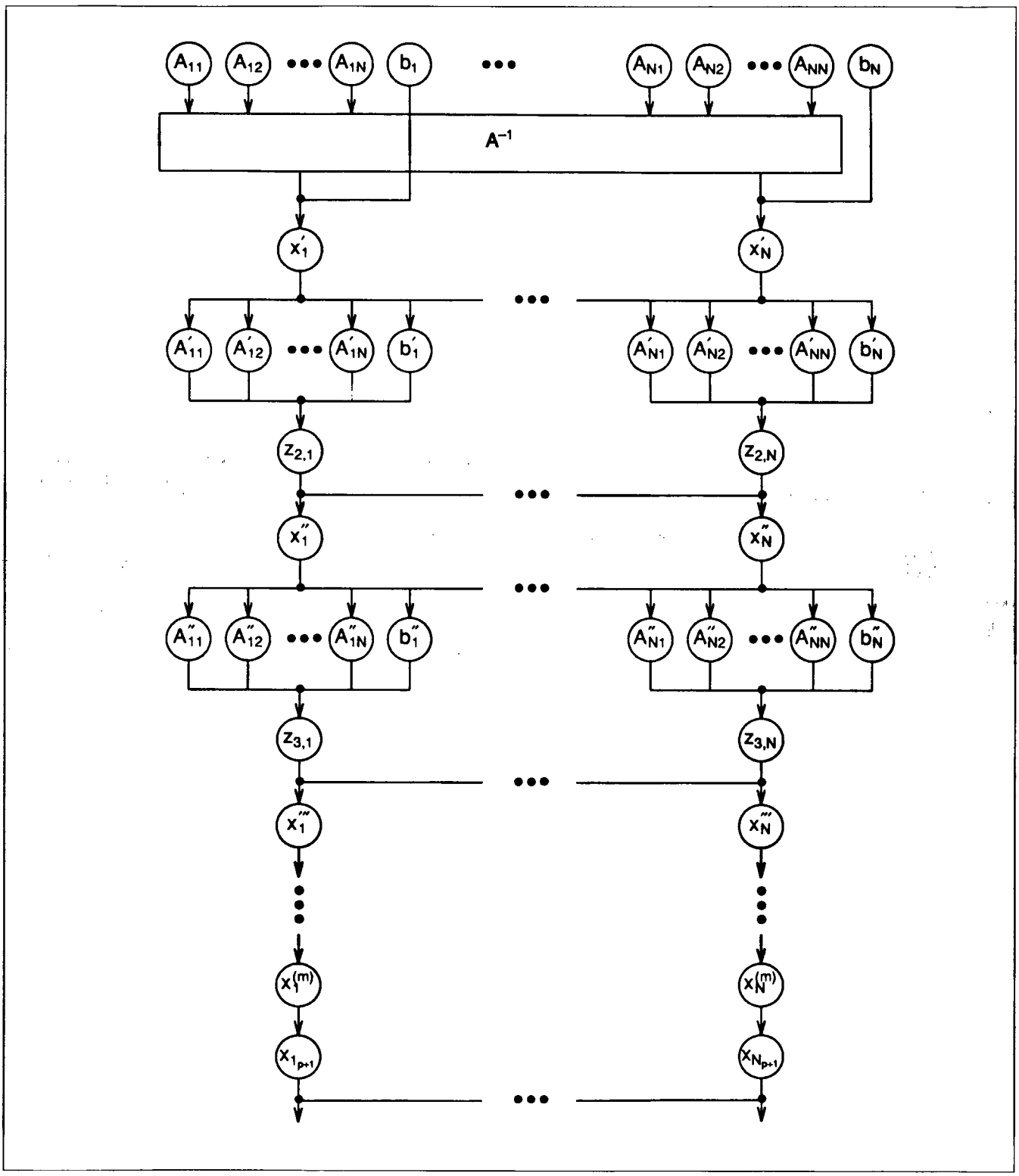



**Figure 1.** Graph of the Solution of $A(x,t)\,x = b(x,t)$ with the single-rate EPSE Algorithm.

different from each other. Where the mathematical model takes the state variable form (1), the nodes representing the processing of the elements of the matrix A and the vector z are eliminated. In the case of multi-rate EPSE algorithm, at each time level some nodes of the graph are eliminated while others represent different computation. Depending on the requirements of the specific application, the computation can be represented either by the graph representing the solution at a point that all the ODEs are integrated or by a graph whose nodes represent the average of the corresponding nodes of the graphs at several points of the solution. Finally, where there exist discontinuities in the model, the nodes of $x_{p+1}$ are repeated until the breakpoint is determined. In all cases, the array structure shown in Figure 1 is preserved.

The structures of the precedence graph for other integration algorithms show an array pattern similar to that of Figure 1. All graphs have N columns but differ in the number of rows. The graphs of the explicit one-step or multi-step single-derivative algorithms have the appearance of subsets of the graph of the EPSE algorithm, corresponding to the first derivative only. This is true both for single and for multi-rate implementations. Integration across discontinuities for multi-step algorithms requires the entire graph to be repeated until the breakpoint has been determined. The structure of the graphs for the extrapolation and the implicit algorithms are identical to the corresponding explicit ones but are repeated a number of times at each time level.

## Functional Characteristics of the Scheduler

The methodology embodied in the scheduler described in this paper consists of a number of software modules shown in Figure 2. The user must specify the mathematical model and the integration algorithm so as to define the computation to be scheduled, the desired performance of the scheduler, as well as speed characteristics of the multiprocessor computer. The procedure includes the functions of:

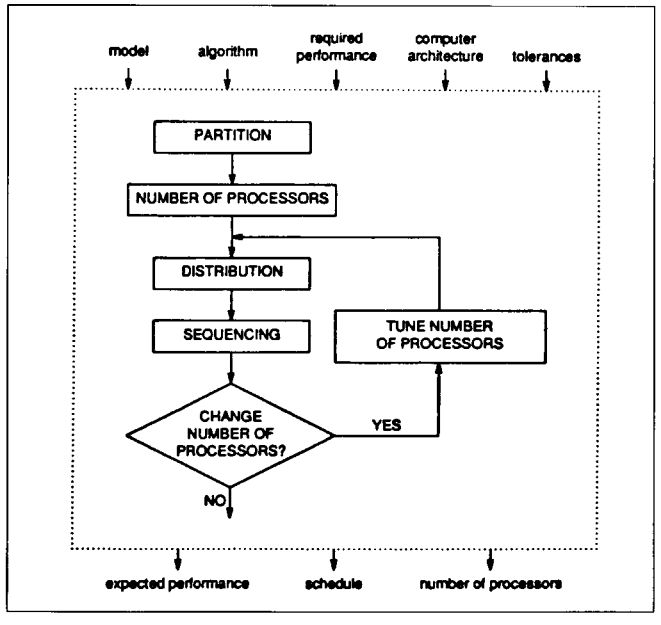



**Figure 2.** High level components of the scheduling procedure.

1. *Partitioning* of the precedence graph applying to the typical point of the solution of a set of ODEs, into a number of loosely-coupled computational tasks.
2. *Determination of the number of processors* by an initial estimation and an iterative tuning to assure the specified performance under minimum cost.
3. *Distribution* of the tasks to the specified number of processors.
4. *Sequencing* of the execution of the tasks to be performed by each processor by specifying the order in which all of the tasks assigned to each processor are to be executed.
5. *Performance prediction* of the schedule computed at each loop.

The output of the procedure includes the number of processors suggested, the schedule and an estimation of the expected performance. If the objective is the minimization of the completion time for a given number of processors, the modules of determining and tuning the number of processors, the module of performance prediction, can be skipped.

## The Partitioning Module

Each of the nodes of the graph of Figure 1 represents a considerable number of arithmetic operations, and the graph is therefore of a relatively high level. The more detailed the graph, *i.e.* the fewer the arithmetic operations encompassed by each node, the greater the flexibility available in scheduling and therefore the more nearly can the optimum be approached. On the other hand, the finer the "granularity" the graph, the more complicated and time consuming the scheduling procedure. From that point of view, the graph shown in Figure 1 constitutes a compromise between optimality and simplicity. The graph has the following characteristics:

1.The graph is deterministic; *i.e.* all 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 operators.
4. The graph has a general precedence structure; *i.e.* each node can have an arbitrary number of predecessors and successors.
5. The times of execution of the nodes are of unequal length.
6. No deadlines exists for individual nodes or for clusters of nodes.

In the design of the scheduling procedure the following assumptions were made:

1. No preemption takes place; *i.e.* a node cannot be interrupted in order to commence the execution of another node with a higher priority. Preemption may increase the speed of the computation but is difficult to implement.
2. All processing elements are identical.
3. Unlimited resources, such as storage or interprocessor communication paths, are available.

The problem of scheduling the execution of a graph, such as Figure 1, is known to be generally NP-complete (Garey & Johnson 1979). NP-completeness implies that an optimal solution is obtained only through the comparison of all of the combinations of nodes, n, and processors, M, requiring of the order of $M^n$ steps. For example, the graph of only 100 differential equations may consist of 120,000 nodes and would require $10^{68000}$ steps to be optimally scheduled on four processors. As a result, a variety of sub-optimal heuristic algorithms for the partitioning of a precedence graph have appeared in the literature (Rammamourthy *et al* 1972, Fernandez & Bussel 1973, Adam *et al* 1974, Kohler 1975, Gonzalez 1977, Makoui 1986, Makoui & Karplus 1987, Ravi 1987). The computational complexity of these algorithms is polynomial with the number of nodes of the graph. Nevertheless, for many practical applications the graphs are so large that these scheduling algorithms may still be too time-consuming ($10^{10}$ steps for a relatively fast $O(n^2)$ algorithm scheduling the solution of 100 ODEs on four processors), so that some kind of clustering of the nodes is required.

For the scheduler described in this paper, the partitioning algorithm performs such a clustering of the nodes of the precedence graph by taking special advantage of the structure of the graphs commonly found in ODE models. The procedure entails the partitioning of the precedence graph into a small number of relatively large autonomous clusters of nodes, called *tasks*, that are tightly coupled to each other and loosely coupled to the nodes comprising the other clusters. This approach is reasonable because the minimization of task interdependence contributes to the minimization of the idle times of the processors as well as to the minimization of the storage requirements, because the data that must be duplicated in different processors are thereby minimized.

Observe that in Figure 1 the nodes of the columns of the graph constitute N loosely-coupled streams. These therefore would appear to be the best candidates for clustering. Since each column of the graph represents the processing of an entire differential equation, or row of the matrix A for the formulation (2), this simple but sufficient partitioning reduces the scheduling problem into the one of scheduling N loosely-coupled tasks on M processors. The scheduling algorithm therefore is polynomial with the number of differential equations, rather than with the number of nodes.

The length of the tasks is critical for the performance of the distribution procedure. In that connection it may occur that the time to complete one or more of these tasks is either longer than the deadline or much longer than the others, resulting in a poorly balanced distribution, as shown in Figure 3, even with an optimal scheduler. The following procedure to improve partitioning is designed to cope with such a situation. The approach is to repartition each of the long tasks into a number, L+1, of subtasks, where L is specified by the distribution procedure.

In order to preserve the structure of the graph, the repartitioning of long tasks is performed as a transformation of the model, by adding L new variables, rather than partitioning the corresponding column of the graph. Figure 4 shows an example of such a transformation of a model in the form (2), with task $T_k$ partitioned into four subtasks. Figure 4a shows the original set of ODEs, and Figure 4b shows the transformed set. Forms (1) and (3) can be transformed similarly.

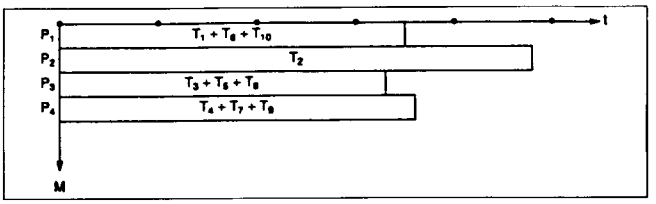



**Figure 3.** Very unequal length tasks may lead to poorly balanced distribution.

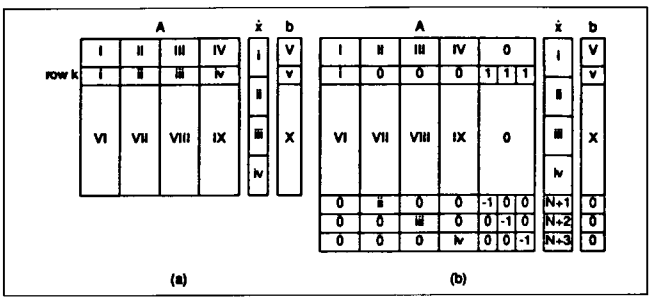



**Figure 4.** Partitioning of the Long Task $T_k$ Into Four Subtasks.

## The Module of Determining the Number of Processors

The objective is to determine the minimum number of processors that can complete the execution of N loosely-coupled tasks by a certain deadline D. Since the completion time, $P_{max}$, decreases monotonically with the number of processors, $M \leq N$, the optimal number of processors, $M_{opt}$, is the maximum integer M that satisfies $P_{max}(M) \leq D$. $M_{opt}$ is determined heuristically because $P_{max}(M)$ is very difficult to express analytically. The optimization of the number of processors represents an attempt to minimize hardware cost and allow sharing the computer by more than one applications.

The approach employed is based on the assumption that the tasks can be treated as independent of each other. This assumption, which constitutes a compromise between optimality and simplicity, is reasonable because in most practical applications the impact on the load balancing of the interprocessor communication, *i.e.* of the idle times of the processors due to the coupling between the equations, is relatively very small. This is due to the followings facts:

1. Interprocessor communication occurs relatively infrequently because the processing time of each task, *i.e.* each equation, is usually fairly long.

2. The number of tasks is usually much larger than the number of processors requiring that many tasks are assigned to each processor. As a result much of the need for interprocessor communication between co-resident tasks is eliminated.

3. As shown in Figure 1, the interprocessor communication serves for synchronization of the tasks. If the processing time of the equations is distributed among the processors in a balanced way, then the communication takes place at the same time for all the processors and as a result it does not destroy the load balancing.

4. It is usually possible to overlap interprocessor communication with arithmetic processing. Then tasks can be arranged to be executed in a sequence that data required by each processor arrive before needed. This is the objective of the sequence procedure discussed below.

5. Much of the loss of accuracy in case of long idle times is compensated by the iterative tuning of the number of processors.

The performance of the scheduling procedure for extensive coupling among the tasks and for long interprocessor communication times is studied in the performance evaluation section.

The problem of determining the minimum number of processors that can execute a set of independent tasks by a certain deadline, can be modeled as a classical bin-packing problem. The FFD (First Fit Decreasing) heuristic (Coffman *et al* 1978, Johnson *et al* 1974), illustrated in Figure 5, has been used successfully to find a quick approximate solution to this problem. The performance of the heuristic is shown in Table 2. The timing diagram of Figure 6 shows the number of processors that the heuristic suggests for ten preordered tasks.

While the FFD heuristic is generally satisfactory, it may occur that the model may be ill-conditioned so that one or more tasks are longer than the deadline, and therefore the problem of optimizing the number of processors has no solution. To cope with such situations, the FFD heuristic is

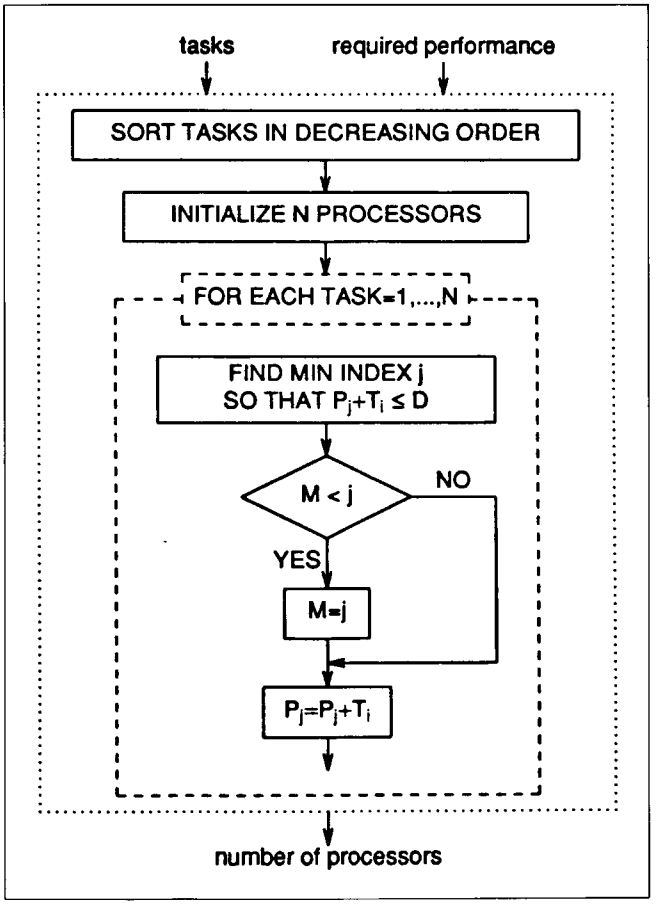



**Figure 5.** The FFD heuristic

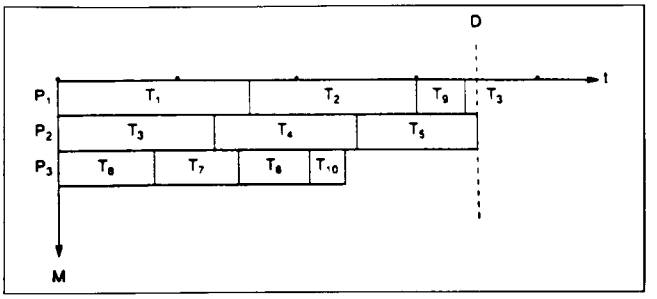



**Figure 6.** An example of FFD determination of number of processors.

**Table 2.** Characteristics of the FFD, LPT and IC heuristics.

| Performance | FFD | LPT | IC |
|---|---|---|---|
| computational complexity | $O(N\log_2 N)$ | $O(N\log_2 N+N\log_2 M+N)$ | $O(N\log_2 M+N)$ |
| worst relative error | $\frac{11}{9}+\frac{4}{M_{opt}}$ | $\frac{M-1}{rM}$ | $\frac{M-1}{rM-M+1}$ |

extended as shown in Figure 7. The long tasks are repartitioned through the transformation of the model into L+ 1 subtasks, L of the subtasks being of length D. Then N tasks are to be distributed among M–L processors. These lengths of the subtasks are chosen in order to reduce the number of processors, since the distribution procedure, as explained in the following section, performs better for fewer processors.

## The Distribution Module

The objective is to distribute N loosely-coupled tasks among M processors in a way that results in a balanced distribution of the load and therefore in earliest completion time. The approach employed is based again on the assump-

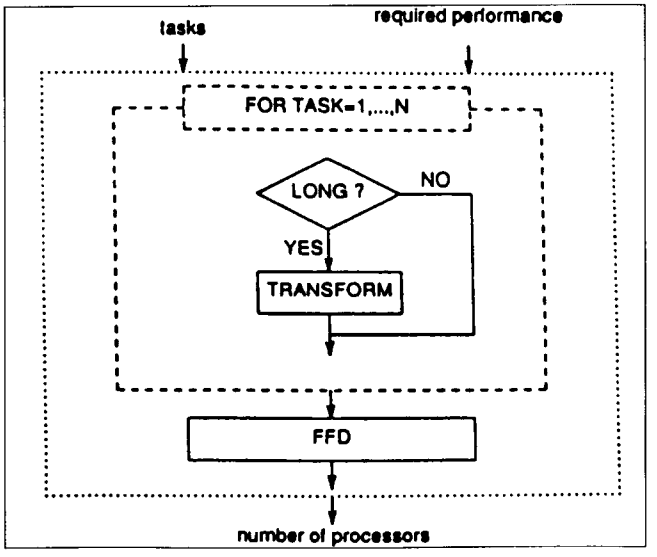



**Figure 7.** Implementation of the Module of determining the number of processors.

tion that the tasks can be treated as independent of each other. The two highest-performance heuristics for scheduling independent tasks are compared in Table 2. The parameter r is the number of tasks distributed to the processor with the longest processing time. It has been selected to be close to N/M, as is to be expected in most cases. The LPT (Longest Processing Time) heuristic, described in [13,4,5], is the closest to optimal. The IC (I/O-Interchange) heuristic, described in [8], is nearly as close to the optimum but relatively faster because it does not require the $O(N\log_2 N)$ steps for sorting the tasks as LPT does. The scheduler presented in this paper is based on the LPT algorithm since the FFD heuristic performed before saves the LPT from sorting the tasks.

The structure of the LPT heuristic is shown in Figure 8, while the timing diagram (known also as *Gantt chart*) of Figure 9 shows the distribution that would be achieved by the LPT heuristic for the ten preordered tasks of Figure 5. Note that since the ODEs of the model are solved in parallel, as shown in Figure 1, the tasks on each processor are executed in parallel, i.e. in an interleaved manner, rather than in sequence as shown in Figure 9.

While the performance of the LPT heuristic is generally satisfactory, on occasion, as that of Figure 3, the relative lengths of the tasks may prevent the reaching of an acceptable balance. In that case, a series of successive improvement steps are taken, as shown Figure 10. These improvements allow infinite approach to a perfect balance of the load among the processors.

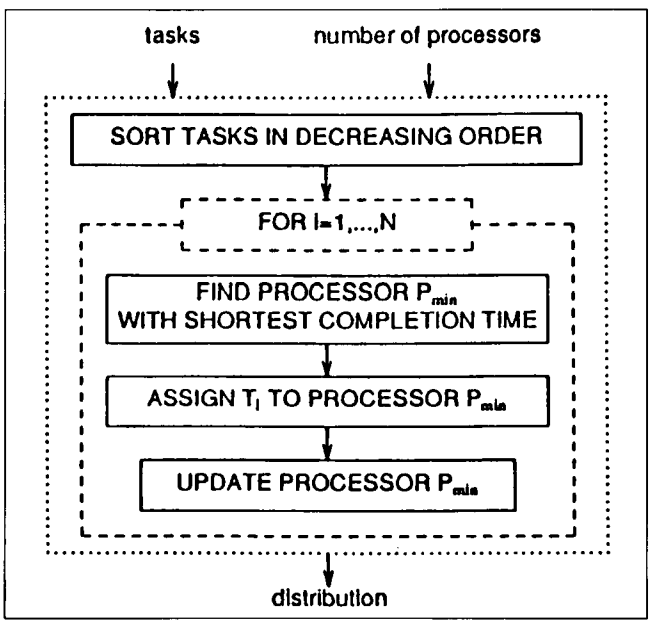



**Figure 8.** The LPT heuristic

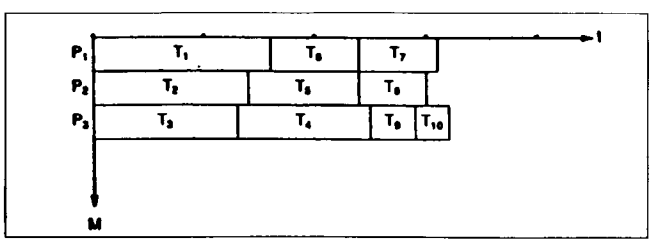



**Figure 9.** An example of a LPT distribution.

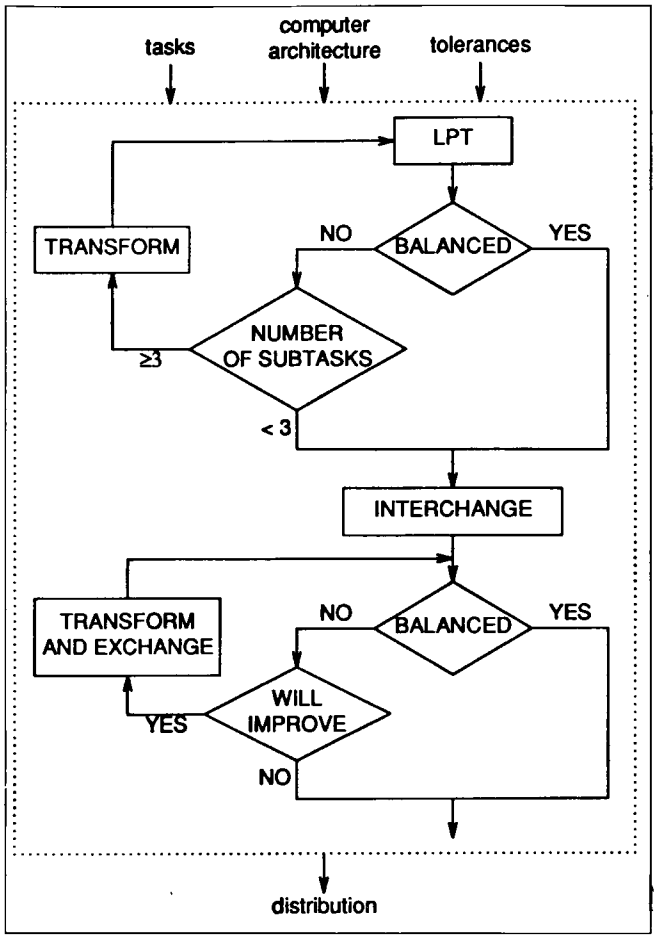



**Figure 10.** Implementation of the distribution module.

It may occur, for example, that one of the tasks is very long, so that the processor, $P_{max}$, to which it is assigned has the longest completion time which is at least twice as long as the completion time of the processor with the shortest completion time, $P_{min}$. This ill-conditioned situation can be eliminated by partitioning the long task into a number of subtasks through a transformation of the model and the reapplication of the LPT heuristic. The subtasks are chosen to be of equal length because LPT performs better for equal length tasks. Then the number of subtasks is selected accordingly to be the smallest integer exceeding $P_{max}/P_{min}$. This iterative process essentially constitutes an improvement of the LPT algorithm.

Also, since the LPT heuristic is suboptimal, an iterative exchange of tasks between processors $P_{max}$ and $P_{min}$ may improve the balance. Figure 11 shows how the INTERCHANGE module performs such a task interchanging. A task T on $P_{max}$ is interchanged with a set of S tasks on $P_{min}$. T and S are selected so that the new completion times are shorter and more balanced. The INTERCHANGE procedure can be viewed as an improvement of the 1/O-Interchange algorithm.

Finally, in case an unacceptable unbalance is still present, a task on processor $P_{max}$ can be iteratively divided into two subtasks, one of which is moved to $P_{min}$. In order to improve balance, the task to be moved has length $1/2\,(P_{max} - P_{min})$. Since the transformation of the model may increase the length of the tasks, the improvement of the balance may result in a longer completion time of the new $P_{max}$ processor. This

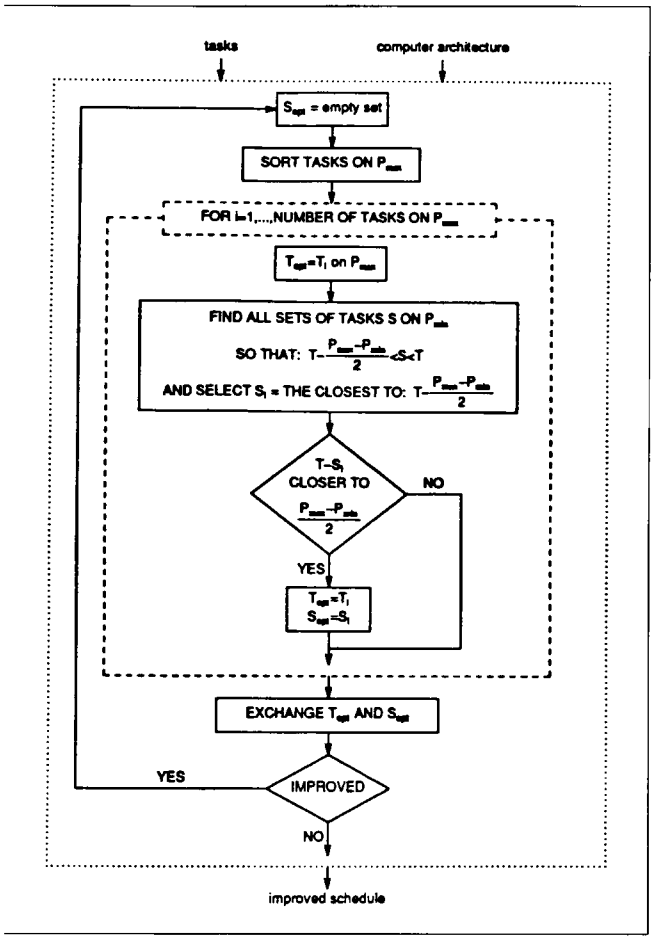



Figure 11. Task interchange to improve the results of the LPT heuristic.

improvement step of the distribution step is therefore repeated only as long as it improves the overall completion time.

## The Sequencing Module

The sequencing module determines the priority of execution of the tasks executed on each processor so that the idle times of the processors are minimized. Optimal sequence maximizes the number of data that arrive before needed. Sequencing tasks is meaningful only in case interprocessor communication can be overlapped with arithmetic processing. The sequencing procedure is shown in Figure 12, in which for each processor the tasks whose results are required by the tasks of the other processors are given a higher priority. Since all of the tasks are executed in parallel in an interleaved manner, this priority is applied to the tasks at every stage of the computation (such as the computation of the same derivative of all the dependent variables).

## The Module of Tuning the Number of Processors

Having selected the schedule for the processors that the extended FFD algorithm suggests, a discrete event simulator can be used to generate the actual timing diagram of the

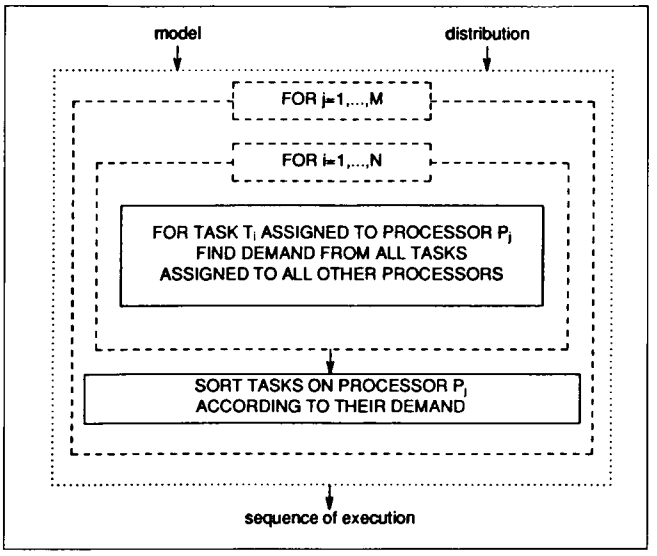



Figure 12. Implementation of the sequencing module.

execution of the tasks showing the busy and idle periods and the actual completion times of the processors. Depending on the performance predicted by the simulator, the number of processors may be tuned iteratively to improve the schedule as shown in Figure 13. This step attempts to compensate the loss of accuracy due to the assumptions and approximations introduced in the different modules. Since the completion time is monotonically decreasing with the number of processors, convergence is guaranteed. Notice that the first iteration requires the measurement of the completion time for one processor only. The module of tuning the number of processors by determining the improved number $M_{k+1}$ from the numbers suggested in the previous two iterations, $M_k$ and $M_{k-1}$, can be implemented as shown in Figure 14.

## Performance Evaluation

The purpose of the procedure described in this paper is to determine the minimum number of processors required to

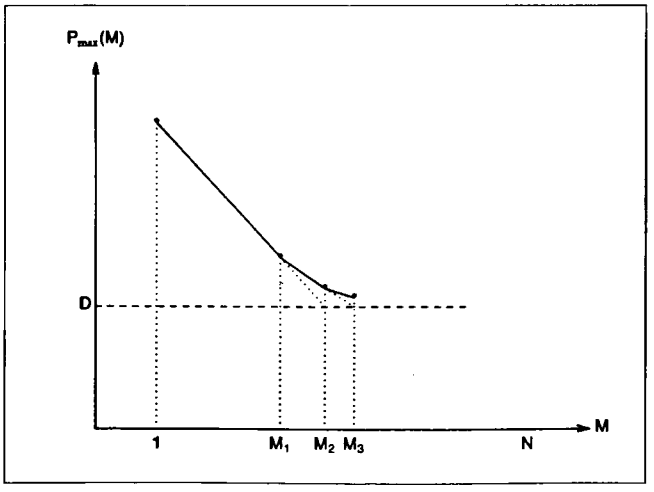



Figure 13. The iterative tuning of the number of processors.

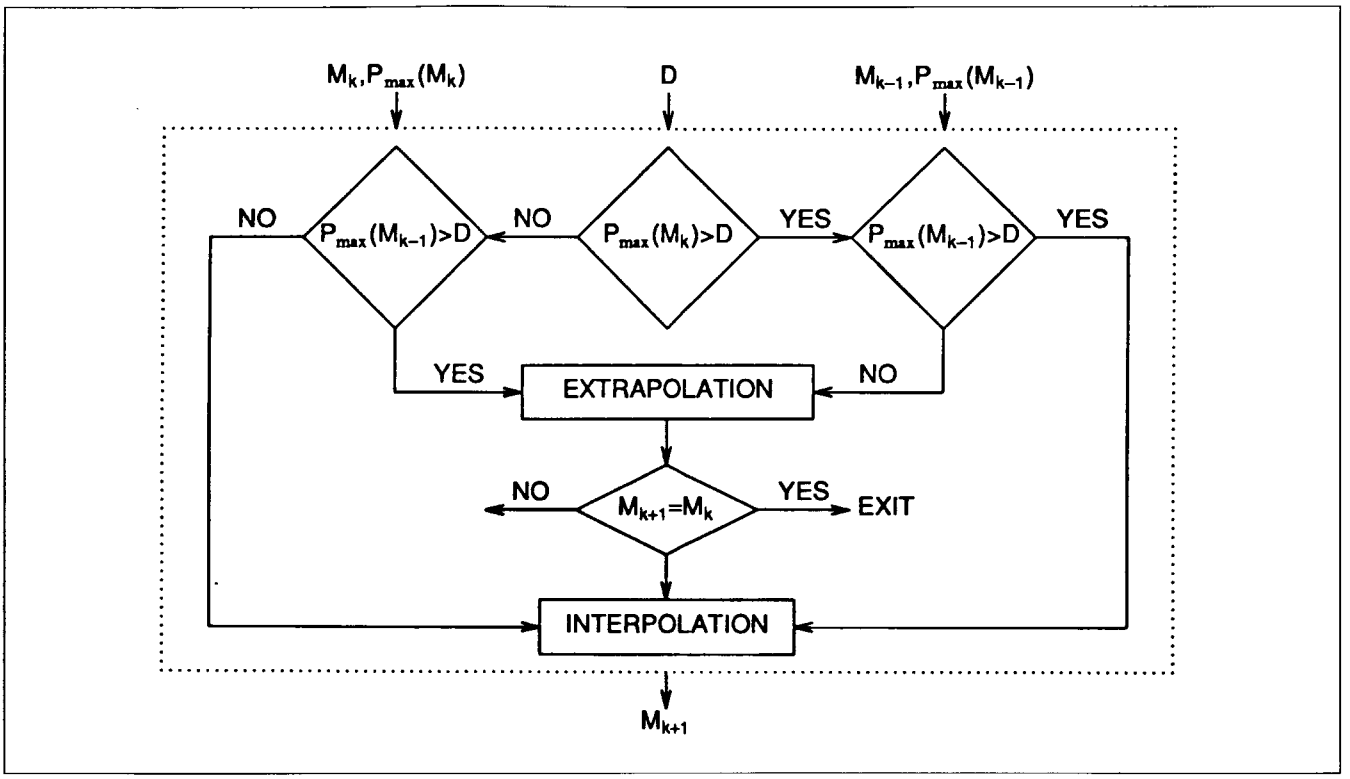



**Figure 14.** Implementation of the module for tuning the number of processors.

satisfy specified deadlines and to schedule the computations to be carried out by each of the processors. An insight into the effectiveness of the procedure can be gained by exploring the speed-up of the solution as the number of processors is increased. The more linear this speed-up, the more effective the scheduling procedure. To this end a series of challenging benchmark studies were carried out. The completion time, *Pmax*, was determined for different parameters. The effect of interprocessor communication delays is of particular interest. Such delays may be the caused by long communication times inherited in the hardware and the communication software of the computer, as well as by the coupling among the ODEs and the nature of the integration algorithm.

Each benchmark consisted of 100 nonlinear ODEs, each of the form $\dot{x} = F(x)$, to be solved in real-time by the single-rate Adams Bashforth algorithm. This algorithm was selected because its data flow graph is a kernel of the graphs of all of the other algorithms and because it is the most widely-used algorithm for real-time applications. The benchmark problems differed in the degree of coupling between the individual equations as well as in the time required for the solution of each equation. The benchmarks included systems of equations that were fully coupled (cp = 100%), partially coupled equations (cp = 50%) and uncoupled (cp = 0%). The task length associated with each equation varied linearly from a minimum value $T_{min}$ for the first equation to a maximum value $T_{max}$ for the last equation, where $T_{max}$ was 100 times longer than $T_{min}$. In general, due to the nature of the LPT and the IC heuristics it is to be expected that the more equal in length the tasks, the more pronounced the speed-up. Therefore, the selected task distribution length constitutes a relatively difficult challenge.

The implementations were programmed for a message-passing multiprocessor. The detailed timing characteristics of the solution of the benchmark problems were computed using emulation models of the multiprocessor computer generated with the aid of the higher-level simulation package NETWORK II.5® (registered trademark and service mark of CACI) (Garrison 1985, Cheung & Karplus 1984, Cheung *et al* 1987). It was assumed that the processing elements are connected in a hypercube topology, and that the interprocessor communication time is proportional to the topological distance between processors. It was also assumed that the computer permits the overlapping of the processing and communications operations on each processing element. The effect of interprocessor communication delays was explored by making the communication time between neighboring processors, *tcom*, $0.1T_{min}$, $T_{min}$, $10T_{min}$ and $100T_{min}$.

The results of the benchmark studies are summarized in Figures 15 and 16. Figure 15 shows the completion time achieved for the benchmarks with 100% coupling as the number of processors is increased from 1 to 100, for communication time *tcom of* $10T_{min}$ and $100T_{min}$. The decision tolerances of the procedure were chosen small enough so that the completion times of the processors were close to the actual deadlines. The plots suggest that the performance of the procedure is satisfactory, since a nearly linear improvement of the completion time is achieved as the number of processors, M, is increased, at least for a relatively small number of processors and relatively short communication times. Communication times of the order of 100 times longer than the shortest tasks dominate the computation and do not allow

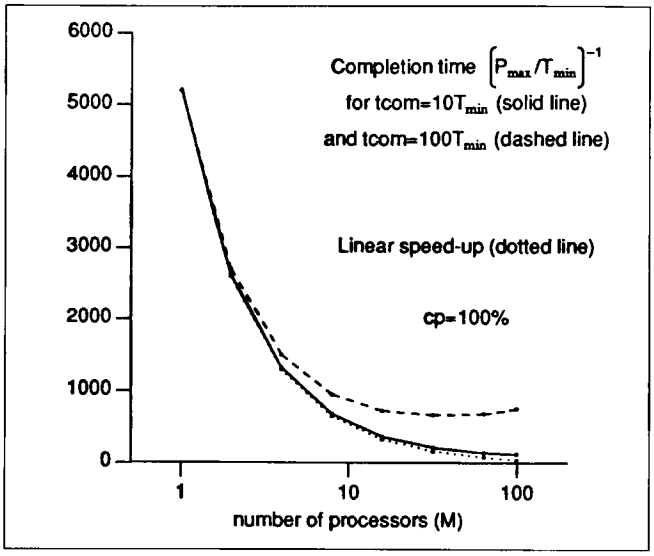



**Figure 15.** The performance of the scheduling procedure.

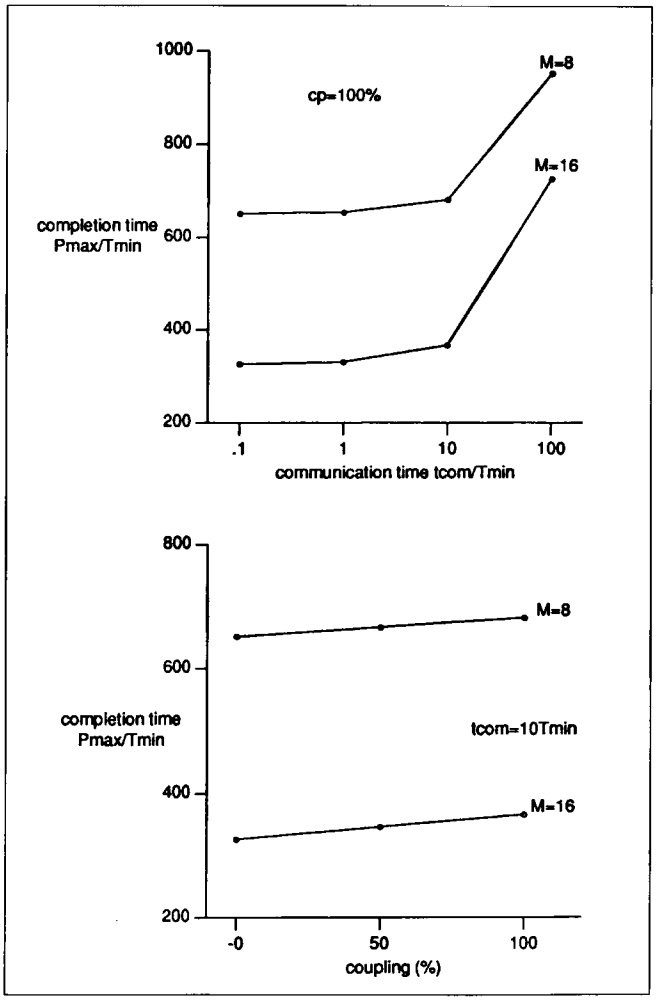



**Figure 16.** The sensitivity of the scheduling procedure against coupling and communication time.

any more the almost linear improvement of the completion time for large number of processors.

Figure 16 shows the effect on the performance of the interprocessor communication time and the coupling for 8 and 16 processors. The plots indicate that the procedure deteriorates almost linearly with the coupling and exponentially with the interprocessor communication time. In addition, the sensitivity of the procedure with the coupling is relatively small. The near-parallelism of the plots confirms the almost linear performance of the procedure for relatively small number of processors. Incidentally, none of the runs reported in Figure 15 and 16 required more than two iterations to tune the number of processors.

## Conclusion

Schedulers are important tools for generating parallel applications for multiprocessor computers. The paper presents a static multiprocessor scheduler for the solution of large sets of ordinary differential equations. The scheduler attempts to fulfill two objectives:

(i) maximum speed of the solution of the model with a given number of processors, and

(ii) maximum speed of the solution of the model with the smallest number of processors that can achieve a specified performance.

The problem of designing such a scheduler is NP-complete hard. It is therefore necessary to employ heuristic techniques and to accept sub-optimal solutions. The procedure described in this paper is designed to combine high performance with simplicity for a wide variety of models, numerical algorithms and multiprocessor architectures. The scheduling is based on the decomposition of the mathematical models as opposed to other approaches that either compile the computation in parallel or schedule the nodes of a data flow graph representation of the computation. The procedure partitions the entire computation into a number of tasks, selects the number of processors, distributes the tasks among the processors and assigns priority of execution of the tasks on each processor. A series of tests on a wide range of benchmarks demonstrated that the procedure achieves an almost linear speedup of the computations.


### Acknowledgment

The studies described in this paper were supported by a joint grant to the Computer Science Department by TRW and by the State of California under the MICRO Electronics Program. The advice and counsel of Dr. R. Gluck of TRW is gratefully acknowledged. The first author wishes to thank also the Onassis and Bodosakis Foundations of Greece as well as the Phi Beta Kappa Association for additional financial support.

## References


Adam, T.L., K.M. Chandy and J.R. Dickson, December 1974. "A comparison of list schedules for parallel processing systems", *Communications of ACM*, v.17, n.12, p.685-690.

Cheung, S. and W.J. Karplus, July 1984. "Tools for simulators consisting of networks of microprocessors", *Proceedings of the Summer Computer Simulation Conference*, Society for Computer Simulation, p.317-325.

Cheung, S., S. Dimitriadis and W.J. Karplus, 1987. *Introduction to Simulation Using NETWORKII.5*, C.A.C.I. Inc., Los Angeles, California.

Coffman, E.G., Jr., 1975. *Computer and Job/Shop Scheduling Theory*, John Wiley & Sons.

Coffman, E.G. and R. Sethi, March 1976. "A generalized bound on LPT sequencing", *Proceedings of the International Symposium on Computer Performance Modeling, Measurement and Evaluation*, Chen, P.S. & Franklin, M.A. editors, p.306-310.

Coffman, E.G., Jr., M.R. Garey and D.S. Johnson, February 1978. "An application of bin-packing to multiprocessor scheduling", *SIAM Journal on Computing*, v.7, n.l, p.1-17.

Femandez, E.B. and B. Bussell, August 1973. "Bounds on the number of processors and time for multiprocessor optimal schedules", *IEEE Transactions on Computers*, v.C-22, n.8, p.745-751.

Finn, G. and E. Horowitz, 1979. "A linear time approximation algorithm for multiprocessor scheduling", *BIT*, v.19, n.3, p.312-320.

Franklin, M.A, May 1978. "Parallel solution of ordinary differential equations", *IEEE Transactions on Computers*, v.C-27, n.S, p.413-420.

Garey, M.R. and D.S. Johnson, 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company.

Garrison, W.J., September 1985. *NETWORK II.5 User's Manual*, C.A.C.I. Inc., Los Angeles, California.

Gonzalez, M.J., September 1977. "Deterministic processor scheduling", *ACM Computing Surveys*, v.9, n.3, p. 173-204.

Graham, R.L., March 1969. "Bounds on multiprocessor timing anomalies", *SIAM Journal on Applied Mathematics*, v. 17, n.2, p.416-429.

Halin, H.J., 1983. "The application of Taylor series methods in simulation", *Proceedings of the Summer Computer Simulation Conference*, p. 1032-1076.

Johnson, D.S, A. Demers, J.D. Ullman, M.R. Garey and R.L. Graham, 1974. "Worst-case performance bounds for simple one-dimensional packing algorithms", *SIAM Journal on Computing*, v.3, p.299-326.

Karplus, W.J., January 1987. *Multiprocessors and Array Processors*, Simulation Series, Society for Computer Simulation, v. 18, n.2.

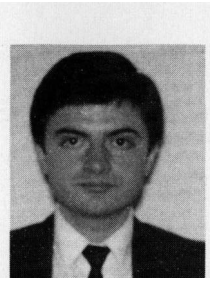
Kohler, W.H., December 1975. "A preliminary evaluation of critical methods for scheduling tasks on multiprocessor systems", *IEEE Transactions on Computers*, v.C-24, n.12, p.1235-1238.

Makoui, A., December 1986. *Data Flow Techniques for Multiprocessor Simulation*, Computer Science Department, University of Califomia at Los Angeles, Tech. Report CSD 860014.

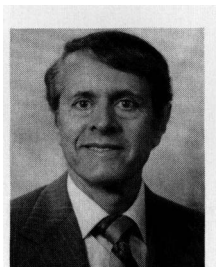
Makoui, A. and W.J. Karplus, August 1987. "ALI: A CSSL/multiprocessor software interface", *SIMULATION*, v.49, n.2, p.63-71.

Rammamourthy, C.V., K.M. Chandy and M.J. Gonzalez, February 1972. "Optimal scheduling strategies in a multiprocessor system", *IEEE Transactions on Computers*, v.C-21, n.2, p.137-146.

Ravi, T.M., M.D. Ercegovac, T. Lang and R.R. Muntz, May 1987. "Static allocation for data flow multiprocessor system", *Proceedings of the Second International Conference on Supercomputing*, v.3, p.169-178.


SOCRATES DIMITRIADIS received a B.S. degree in Electrical Engineering from the Aristotelian University of Thessaloniki, Greece in 1981, and a M.S. and a Ph.D degree from the University of California at Los Angeles in 1983 and 1986 respectively. He was also a postdoctoral researcher at the University of Califomia until June 1988, performing research on distributed simulation of the dynamics of space structures for the Space & Technology Group of TRW. He is currently working at Teledyne Controls on a real time multiprocessor system for flight testing and as an independent technical consultant on numerical computing, distributed processing and simulation. He is the coauthor of the book: *Introduction to Simulation using NETWORK II.5* (CACI 1987). He has been awarded honors and scholarships by several Greek and American Institutions. He is a member of the Eta Kappa Nu Association, the Phi Beta Kappa Association and the Technical Chamber of Greece. His current interests include multiprocessing, CAD/CAM, computer communications, and computer applications for developing countries.


**WALTER J. KARPLUS received his BS (1949) in Electrical Engineering from Cornell University, his MS (1951) in electrical engineering from the University of California, Berkeley, and his PhD (1955) in Engineering from the University of California, Los Angeles.**

**Dr. Karplus is a faculty member of the School of Engineering and Applied Science at UCLA, where he is a professor and former chairman of the Computer Science Department. He has taught and directed research in computer methodology, on-line computing, and computer simulation. He is also the head of the Center for Experimental Computer Science at UCLA. Dr. Karplus has held engineering positions with the Hughes Aircraft Company, International Geophysics Company, and Sun Oil Company. He is the author, co-author, or editor of ten books and has published more than 100 technical papers in computer and electrical engineering fields. He holds four patents and is a registered Electrical Engineer in California.**

**Dr. Karplus served as President of The Society for Computer Simulation from 1982-1984.**