NASA Technical Memorandum 89850

Time-Partitioning Simulation Models for Calculation on Parallel Computers

Edward J. Milner, Richard A. Blech, and Rodrick V. Chima
Lewis Research Center
Cleveland, Ohio

Prepared for the
1987 Summer Computer Simulation Conference
sponsored by the Society for Computer Simulation
Montreal, Canada, July 27–30, 1987

NASA

TIME-PARTITIONING STIMULATION MODELS FOR CALCULATION ON PARALLEL COMPUTERS

Edward J. Milner, Richard A. Blech, and Rodrick V. Chima
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135

SUMMARY

A technique allowing time-staggered solution of partial differential equations is presented in this report. Using this technique, called time-partitioning, simulation execution speedup is proportional to the number of processors used because all processors operate simultaneously, with each updating the solution grid at a different time point. The technique is limited by neither the number of processors available nor by the dimension of the solution grid. Time-partitioning was used to obtain the flow pattern through a cascade of airfoils, modeled by the Euler partial differential equations. An execution speedup factor of 1.77 was achieved using a two processor Cray X-MP/24 computer.

INTRODUCTION

The trend in aeropropulsion system designs has been to try to obtain more and more power with less and less weight. To achieve this, simplicity is generally sacrificed in order to achieve the increase in performance. Aeropropulsion systems, and their components, become more complex with each new design.

Computational Fluid Dynamics (CFD) is playing an increasingly important role in the design of aeropropulsion systems. This is due to: (1) the high cost of building hardware; (2) the time and expense required to conduct wind tunnel tests of new designs; (3) the lack of facilities to realistically test new designs (testing the National Aerospace Plane concepts at hypersonic speeds, for instance); (4) advances in computational technology; (5) increased understanding of fundamental physics.

The objective of CFD is to build an understanding of these advanced systems into mathematical models which accurately represent the complicated physics taking place in these systems. The good news is that these mathematical models are evolving through intensive research (both experimental and analytical), but the bad news is that the models are so detailed that time-accurate solutions cannot currently be obtained in a reasonable amount of time. Holst (ref. 1) projects that direct simulation of a Navier-Stokes airfoil simulation using no simplifying assumptions would require approximately 10^{16} computer operations. This amounts to 4 months cpu-time using a state-of-the-art gigaflop computer such as the Cray-2. If solutions were available in minutes or hours, optimized designs could be generated on the computer. Therefore, orders of magnitude increases in computing speed are needed to make CFD practical for aeropropulsion design optimization.

It is generally recognized that computers are fast approaching speed limits. As a result, the 1980's has seen a growing interest in combining state-of-the-art hardware with new architectures and software techniques to

try to achieve the required speedup. Approaches have included vector processing (single instruction-multiple data), multiprocessing (multiple instruction-multiple data), and data-driven architectures. Williams and Bobrowicz (ref. 2) indicate speedup rates of ten or more can be attained combining vector processing with multiprocessing.

Today, almost all supercomputers use vector processing and several (e.g., Cyber 205, Cray X-MP, Cray-2) use multiple vector processors. Programming these supercomputers involves the use of vectorizing compilers that convert source codes, originally intended for conventional single, scalar processor computers, to codes that run efficiently on the vector processors. While today's supercomputers represent a step in the right direction, they still offer only a fraction of the needed computing power because of the limited number of processors (4 or less) and the limited capabilities of the software.

Data-driven approaches to parallel processing have been proposed (refs. 3 and 4) that involve large numbers (hundreds or thousands) of processors. In these cases, calculations are assigned to processors on a single operation basis. Hundreds of processors could be used to achieve significant speedups in simulations where like numbers of individual operations can be simultaneously carried out. However, software is required to control these calculations and to assign the operations to the processors. The sequencing of hundreds of computers is a tremendous software task.

It seems clear that tapping the tremendous potential of parallel processing will depend upon advancements in software technology. In particular, software needs to be developed which can automatically map complex, multi-dimensioned codes onto parallel architectures, making effective use of available scalar and vector processing resources.

Researchers at NASA Lewis Research Center are actively engaged in a research program (refs. 5 to 14) to explore parallel processing techniques for analyzing internal flows in aeropropulsion systems. One of the objectives of that research is to identify parallel architectures and algorithms that are well suited for three-dimensional Navier-Stokes flow solvers. Another objective is to devise techniques for effectively partitioning the solver calculations for parallel solution.

This paper discusses a partitioning technique which allows calculations at the next time interval to begin before all calculations at the current time interval are completed. The authors refer to the method as time-partitioning. The next section of this report discusses time-partitioning and other partitioning methods, pointing out the advantages and disadvantages of each. Time-partitioning is then applied to a restriction in a flow field problem. This example represents an important class of problems relating to computing flows in turbomachinery cascades. The resulting speedup, obtained using the Cray X-MP, is discussed, as well as the steps required to develop and implement the time-partitioned simulation.

PARTITIONING METHODS

Partitioning the simulation into work units and allocating those work units to processors (packing) is one of the most difficult tasks which must be addressed in parallel processing. The way that the simulation is partitioned

and packed directly affects the speed at which the simulation executes and the efficiency of processor use. Work loads should be balanced among the processors to eliminate excess processor idle time. The level of parallelism being considered greatly affects the ease with which processor work load balance is achieved, as shown in table I. This table summarizes key characteristics of three partitioning methods about to be discussed.

Data-driven architectures consider parallelism in a simulation at its most basic operational level. An operation is considered the basic unit of work. When an operation is triggered, a processor is assigned by the system to carry out that operation. When that single calculation is completed, the processor is free to be assigned to another waiting calculation. In this case, then, processor load is very simply a single operation. Processor assignment, on the other hand, is very difficult. In a large simulation, literally hundreds of additions and multiplications may be ready to be carried out simultaneously. Processors to service them are normally assigned on the fly while the simulation is executing. The bookkeeping for tracking which processors are currently busy and which are available for assignment is tremendous. Sophisticated software is required to manage this task.

Assigning the equation as the basic unit of work eliminates the requirement of having to assign processors on the fly to carry out parallel calculations. Equations are assigned for computation to the processors before execution begins. Relatively few processors are required to execute a simulation. (The helicopter engine simulation of reference 10 required only six to achieve minimum execution time.) However, using the equation as a basic unit of work makes this architecture one level removed from the data-driven architectures. Whereas before, work balance on the processors was no consideration, now it is an important consideration. Depending on the complexity of the equation, the time to calculate the output of an equation will vary. Hence, work load balance among the processors cannot be achieved by just assigning an equal number of equations to each processor. Equation execution times must be determined and sequential calculation paths must be identified. The longest such path is designated the critical path because its execution time is the minimum possible execution time of the simulation. To achieve this minimum execution time the critical path equations must reside on a processor by themselves. The other paths must be packed on remaining available processors in a way that the execution time of no processor exceeds that of the critical path processor. The entire process of partitioning and packing mathematical models for parallel calculation has been automated. Reference 15 discusses the procedure in detail; block diagrams of the process are included in the report.

Each of the partitioning methods discussed above performs all calculations within the same time interval. Time-partitioning, as the name implies, performs calculations at different time intervals simultaneously. This partitioning technique is particularly suited for problems requiring solution of partial differential equations over a grid.

Whereas other partitioning methods identify vectorization as scalar parallelism, time-partitioning maintains vectorization within the grid calculations.

For simplicity, the discussion here will assume a two-dimensional grid. The assumption is made for convenience only in descibing the time-partitioning concept and does not imply limitations on the generality of the method. The

concept readily extends to n-dimensional grids. However, time-partitioning does require that current state variable values be dependent only on neighboring-node past values. This condition would be met if an explicit integration method were used, for example.

Time-partitioning can be used to reduce the effective calculation time of the simulation if parameter update calculations over the grid are completed in the systematic fashion described in the following paragraph. At some point, before all grid node updates for the current time interval have been completed, sufficient information will be available to begin updating grid nodes at the next time interval.

Suppose that the current parameter values at each node of a rectangular grid are dependent only on past values at two columns of neighboring nodes. If the grid nodes are updated column-wise from left to right, once three columns of nodes have been updated, sufficient information exists to begin updating the leftmost node columns at the second time interval. Since the same kinds of calculations are taking place at each node, calculation time at each node is comparable. Theoretically, then, the first processor set should remain a fixed distance (that is, three columns of nodes) ahead of the second processor set. Hence, there should be no delays caused by the second processor set having to wait for required information from the first.

Likewise, once the second processor set has updated three columns of nodes, sufficient information again exists to begin updating the simulation at the third time interval. This process can continue until all processor sets available are being used or until the first processor set has completed its time interval update. In either case, the first processor set will update the next time interval. The process continues to repeat until the simulation run has been completed. A diagram of the time-partitioning execution process is shown in figure 1.

An outstanding feature of time-partitioning is the ease with which the technique can be implemented. Basically the same equations are executing on each processor, but at different time points. Because of this, the processor work load is almost naturally balanced. The process can be implemented with as little as two processors, and the theoretical speedup factor realized is proportional to the number of processors used. Processor idle time is virtually nil.

To use time-partitioning techniques requires that parameters at a node can be updated using only past values of parameters at some level of neighboring nodes. Thus, a simulation using an implicit integration method could not be time-partitioned due to the iterative nature of the solution and the interdependence of the parameter current values. Time-partitioning techniques were applied to a fluid-flow problem at Lewis Research Center using the two processor Cray X-MP computer. The example problem used and the results obtained are discussed in the following sections.

TIME-PARTITIONED SIMULATION DEVELOPMENT

An important class of fluid flow problem deals with computing flows in turbomachinery cascades. This flow information is vital to developing efficient new turbomachinery designs. Calculating the flow about the cascade of

bicircular arc airfoils shown in figure 2 is representative of this class of problems, and is well-documented by Johnson and Chima (refs. 16 to 19).

The cascade of airfoils can be used to model many different systems. For instance, two adjacent airfoils could model the convergent-divergent nozzle of a jet engine.

Chima and Johnson model the cascade of airfoils using the thin-layer version of the Navier-Stokes equations. The thin-layer assumption is implemented by using a body-fitting coordinate system and neglecting the viscous terms in the coordinate direction along the body. Initial conditions are specified as uniform flow at the isentropic Mach number implied by the ratio of exit static pressure to inlet total pressure. Specified inlet boundary conditions are total pressure, total temperature, and flow angle; at the exit, static pressure is specified. For inviscid flow, the tangency condition is applied along solid surfaces as shown in figure 2. Starting with a 65 by 17 grid and using the multi-grid acceleration scheme discussed in reference 17, Chima and Johnson achieved work reduction factors for inviscid flow calculations ranging from 1.14 (for choked flow conditions at Mach 0.73) to 4.02 (for low-speed flow at Mach 0.2). For Mach 0.5, a work reduction factor of 3.31 was achieved.

Time-partitioning techniques were applied to the cascade of airfoils problem for three reasons. First, as was mentioned above, it is an important problem in computational fluid mechanics. Results obtained are important in designing components which are more efficient than those currently available.

A second reason is that the Chima-Johnson multi-grid simulation could be used as a standard for verifying the results coming from the time-partitioned model being developed. A valid time-partitioned simulation would produce results consistent with those from the multi-grid simulation.

And finally, the multi-grid simulation could be used as a basis for developing the time-partitioned model. Chima and Johnson use a second order Runge-Kutta integration update. This is an explicit integration technique requiring only past values to update state variables. This lends itself very nicely to time-partitioning.

This time-partitioning study was carried out using a 33 by 9 grid (fig. 3) at Mach 0.5 conditions. Computations are made column-wise in the discussion which follows, although the grid can actually be updated either row-wise or column-wise. The former vectorizes a row of length 33 as opposed to a column of length 9; as shown later in the COMPUTATIONAL RESULTS section, the latter allows use of up to eleven processors as opposed to three.

The Chima-Johnson simulation was written with the intent of executing the code on a single, serial processor. Because of this, considerable reorganization of the simulation was required in order to make it conducive to parallel computation and time-partitioning techniques.

Reorganizing the Chima-Johnson simulation to meet time-partitioning needs required considerable care. The simulation had been designed to carry out calculations, a portion at a time, over all nodes of the grid, one set of calculations being completed before the next set would begin. Time-partitioning requires that a column of nodes be updated completely before beginning the next column of nodes. Effecting this change was not straight forward.

In a simulation coded to execute on a single, serial processor, memory locations designated to hold updated parameter values can be used as scratch memory to hold intermediate values for other calculations before those parameters are updated. However, when time-partitioning is being used, memory cannot generally be used for dual roles. Once a column of nodes is updated, their values must not be changed because those values are required for use almost immediately by another processor performing calculations at the next update interval. Calculations in the time-partitioned simulation were arranged as a task that updated a column of nodes every time the task was called. Successive calls to the task updated the nodes a column at a time from left to right across the grid. Once a node column was updated, including its boundary values, no parameter value was changed until the node column was updated again at the next time interval.

Normally, local variables use the same memory locations throughout execution of a simulation. However, to use both Cray X-MP processors simultaneously requires that the program code execute in stack mode. In this mode, local variables are not saved between subroutine calls. Every time that the stack is accessed, different memory locations can be used for holding values of the local variable. A variable required to maintain its value between subroutine calls must be a global variable. Care must be taken not to use local variables as counters, flags, or storage locations for needed information at a subsequent time. One way of ensuring global status is to include the variable in a COMMON statement.

The reorganized simulation was validated by executing it on a single processor in stack mode on the Cray X-MP. A steady-state solution was obtained in 5.47 sec execution time and the results agreed with those from the Chima-Johnson simulation. One thousand eight hundred and seventy calculation cycles were performed and residuals were less than 4×10^{-11}. Residuals are a measure of the maximum differences between successive values of the state variables as simulation execution progresses. As the simulation approaches steady-state conditions, the residuals approach zero. Chima and Johnson use residuals to determine when the solution has converged (ref. 17).

TIME-PARTITIONED SIMULATION EXECUTION

As discussed above, the reorganized simulation was arranged as a task which updated a column of nodes. Successive calls to the task updated node columns from left to right across the grid. To execute the simulation in time-partitioned mode, a duplicate copy of the task code is required. Designate these copies as Task 1 and Task 2 to distinguish them; however, they are identical. Each updates a column of nodes from left to right across the grid with each successive call to that task. Processor set 1 always executes Task 1, and Processor set 2 always executes Task 2. The time-partitioned simulation is executed on the Cray X-MP in the following manner. Task 1 is called three successive times without calling Task 2. This updates the first three columns of nodes at time interval 1, providing sufficient information to begin updating the grid at time interval 2. Hence, on the fourth call to Task 1, and on every call thereafter, a call is also made to Task 2. Thus, Processor set 1 is updating the grid at odd multiples of time, while Processor set 2 is simultaneously updating the grid at even multiples of time. The reason Task 2 must lag Task 1 by three columns of nodes is that the solution finite-difference

scheme uses second-order central differences for the fluxes and a fourth-difference (5 point) artificial viscosity operator for damping.

Since each of the tasks is updating a column of nodes with each task call, they should also complete their respective column calculations at about the same time. To maintain control of the simulation, however, task wait mechanisms are incorporated into the code. This ensures that, as new calls to the two tasks are made, they begin executing simultaneously. Hence, Task 2 is guaranteed to be lagging Task 1 by precisely three columns of nodes.

By not using a task wait mechanism, the programmer would relinquish control of the simulation. If both processor sets were freewheeling - that is, executing their tasks independently and as quickly as possible, Task 2 could actually end up leading Task 1 by the end of the simulation run. For example, a system interrupt to Processor set 1 could momentarily delay its calculations. Task 2 would then be using data in its calculations which had not been updated by Task 1.

COMPUTATIONAL RESULTS

For this initial study of time-partitioning, only Euler equations governing inviscid flow have been considered. However, time-partitioning techniques are also applicable to Navier-Stokes equations governing viscous flow. Typical results obtained from the simulation are the isomachs shown in figure 4. These results were obtained for Mach 0.5 flow conditions. The lines of constant Mach number form a profile of steady-state Mach number within the computational element (figure 2) for these flow conditions. The elapsed execution time of the simulation, however, is what is important for this report.

As shown in table II, the two processor time-partitioned simulation achieved steady-state conditions in 3.09 sec. A total of 1870 calculation cycles were performed, and residuals were less than 4×10^{-11}. Using the time-partitioning techniques, an effective speedup factor of (5.47/3.09=) 1.77 was realized. This represents an efficiency of 89 percent with respect to the theoretical speedup factor of almost two (2 minus time to start the process). This is consistent with the Cray X-MP multitasking overhead reported by Chen (ref. 20). The speedup factor is significant insofar as if more processors were available, a third task could have been set up to begin executing the third time interval on the fourth call to Task 2, etc. Since a new task (and time interval calculation) could begin every time three columns of nodes were calculated, a total of eleven (that is, $33 \div 3$) processors could have been used in the solution on this example problem, with a theoretical speedup in solution time proportional to the number of processors used! (Notice that if calculations were made row-wise instead of column-wise, only three processors could have been used.)

More investigations of time-partitioning will be required in order to answer queries raised by this initial study. Foremost, is determining how using additional processors affects the execution speedup factor obtained. Living in the real world that we do, attaining the predicted theorectical linear relationship hardly seems realistic. Using two processors, the speedup fell about 11 percent short. It is reasonable to assume that the task wait mechanism incorporated into the code to maintain control of the simulation accounts for at least a part of that difference. How the task wait will

affect the execution time when three, four, eight, or more processors are used is something that will have to be investigated. Moreover, the speedup factor obtained using time-partitioning techniques theoretically should not depend heavily on the code or the size of the tasks to be executed. Whether time savings duplicate those obtained in this study when time-partitioning techniques are applied to other codes and other applications is a question that must be investigated. This initial study has given some encouraging results. Time-partitioning shows potential for being a powerful parallel processing tool. Only through further investigation will its effectiveness be determined.

CONCLUDING REMARKS

Parallel processing promises to be a very effective tool for reducing wallclock execution time for many complex simulations.

Time-partitioning techniques discussed in this report provide a means for solving systems of Euler and Navier-Stokes equations at several different time-steps simultaneously. The calculations take place in a time-staggered fashion across the solution grid.

Time-partitioning techniques were used to determine the steady-state flow pattern through a cascade of airfoils. This important computational fluid mechanics problem is characterized by a set of Navier-Stokes partial differential equations. Solution was over a two-dimentional grid using a second-order Runge-Kutta integration. An execution speedup factor of 1.77 was achieved, using the two processors of the Lewis Research Center Cray X-MP computer. Results from this initial study are encouraging. Time-partitioning has the potential for providing an easy means of parallelizing explicit codes and obtaining execution speedup factors proportional to the number of processors used.

The application of time-partitioning techniques is not limited to a particular number of processors. All processors available can be used.

Further studies are required to investigate the relationship between the execution speedup factor achieved and the number of processors used. Time-partitioning should be applied to a variety of codes from different applications. Also, a computer system having at least four processors (preferably more) should be used for that investigation.

The authors welcome discussions of the techniques presented in the paper, related techniques, and developments in the many other aspects of multiprocessor simulation.

REFERENCES

1. Holst, Terry L.: "Numerical Solution of the Navier-Stokes Equations About Three-Dimensional Configurations - A Survey," Supercomputing in Aerospace. NASA CP-2454, pp. 281-289, 1987.

2. Williams, Elizabeth and Bobrowicz, Frank: "Speedup Predictions for Large Scientific Parallel Programs on Cray X-MP-Like Architectures," Proceedings of the 1985 International Conference on Parallel Processing, D. Degroot, ed., IEEE, Piscataway, NJ, pp. 541-543, August 1985.

3. Srini, V.P.: "An Architectural Comparison of Dataflow Systems," IEEE Computer Magazine, Vol. 19, No. 3, pp. 68-88, March 1986.

4. Jamieson, Leah H. and Ashcroft, Edward A.: "Performance Analysis of Data-flow Signal Processing Algorithms," Proceedings of the 1986 International Conference on Parallel Processing, K. Hwang, S.M. Jacobs, and E.E. Swartzlander, eds., IEEE, Piscataway, NJ, pp. 608-610, August 1986.

5. Blech, Richard A. and Arpasi, Dale J.: An Approach to Real-Time Simulation Using Parallel Processing. NASA TM-81731, 1981.

6. Milner, Edward J.: A Generalized Memory Test Algorithm. NASA TM-82874, 1982.

7. Blech, Richard A. and Arpasi, Dale J.: Hardware for a Real-Time Multi-processor Simulator. NASA TM-83805, 1984.

8. Blech, Richard A. and Williams, Anthony D.: Hardware Configuration for a Real-Time Multiprocessor Simulator. NASA TM-88802, 1986.

9. Milner, Edward J. and Arpasi, Dale J.: Simulating a Small Turboshaft Engine in a Real-Time Multiprocessor Simulator (RTMPS) Environment. NASA TM-87216, 1986.

10. Arpasi, Dale J.: RTMPL - A Structured Programming and Documentation Utility for Real-Time Multiprocessor Simulations. NASA TM-83606, 1984.

11. Arpasi, Dale J.: Real-Time Multiprocessor Programming Language (RTMPL): Users Manual. NASA TP-2422, 1985.

12. Cole, Gary L.: Operating System for a Real-Time Multiprocessor Propulsion System Simulator. NASA TM-83605, 1984.

13. Cole, Gary L.: Operating System for a Real-Time Multiprocessor Propulsion System Simulator Users Manual. NASA TP-2426, 1985.

14. Jones, William H.: Increasing Processor Utilization During Parallel Computation Rundown. NASA TM-87349, 1986.

15. Arpasi, Dale J. and Milner, Edward J.: Partitioning and Packing Mathemati-cal Simulation Models for Calculation on Parallel Computers. NASA TM-87170, 1986.

16. Johnson, G.M.: Multiple-Grid Acceleration of Lax-Wendroff Algorithms. NASA TM-82843, 1982.

17. Chima, R.V. and Johnson, G.M.: Efficient Solution of the Euler and Navier-Stokes Equations with a Vectorized Multiple-Grid Algorithm. NASA TM-83376, 1983.

18. Swisshelm, Julie M., Johnson, Gary M., and Kumar, Swarn P.: "Parallel Computation of Euler and Navier-Stokes Flows," Applied Mathematics and Computation, Vol. 19, Nos. 1-4, pp. 321-331, July 1986.

19. Johnson, Gary M., Swisshelm, Julie M., Pryor, Daniel V., and Ziebarth, John P.: Multitasked Embedded Multigrid for Three-Dimensional Flow Simulation. ICS-TR-86004, Institute for Computational Studies, Fort Collins, CO, 1986.

20. Chen, Steve S., Dongarra, Jack J., and Hsiung, Christopher C.: "Multiprocessing Linear Algebra Algorithms on the Cray X-MP-2: Experiences with Small Granularity," Journal of Parallel and Distributed Computing, Vol. 1, No. 1, pp. 22-31, August 1984.

TABLE I. - CHARACTERISTICS OF SOLUTION METHODS

Solution method	Processor assignment	Basic unit of work	Work load balance	Processor idle time	Processors required
Data driven	Difficult, assigned during execution	Single operation	Natural balance	None	Many
Equation driven	Pre-assigned	Single equation	Diffiicult; requires packing	Depends on packing	FEW
Time partition	Pre-assigned	Set of equations	Easily balanced	Virtually none	FEW

TABLE II. - COMPARISON OF SIMULATION EXECUTION

Simulation type	Processors used	Calculation cycles executed	Execution time, sec	Speedup factor	Efficiency, percent
Serial	1	1870	5.47	1.00	100
Time-partitioned	2	1870	3.09	1.77	89

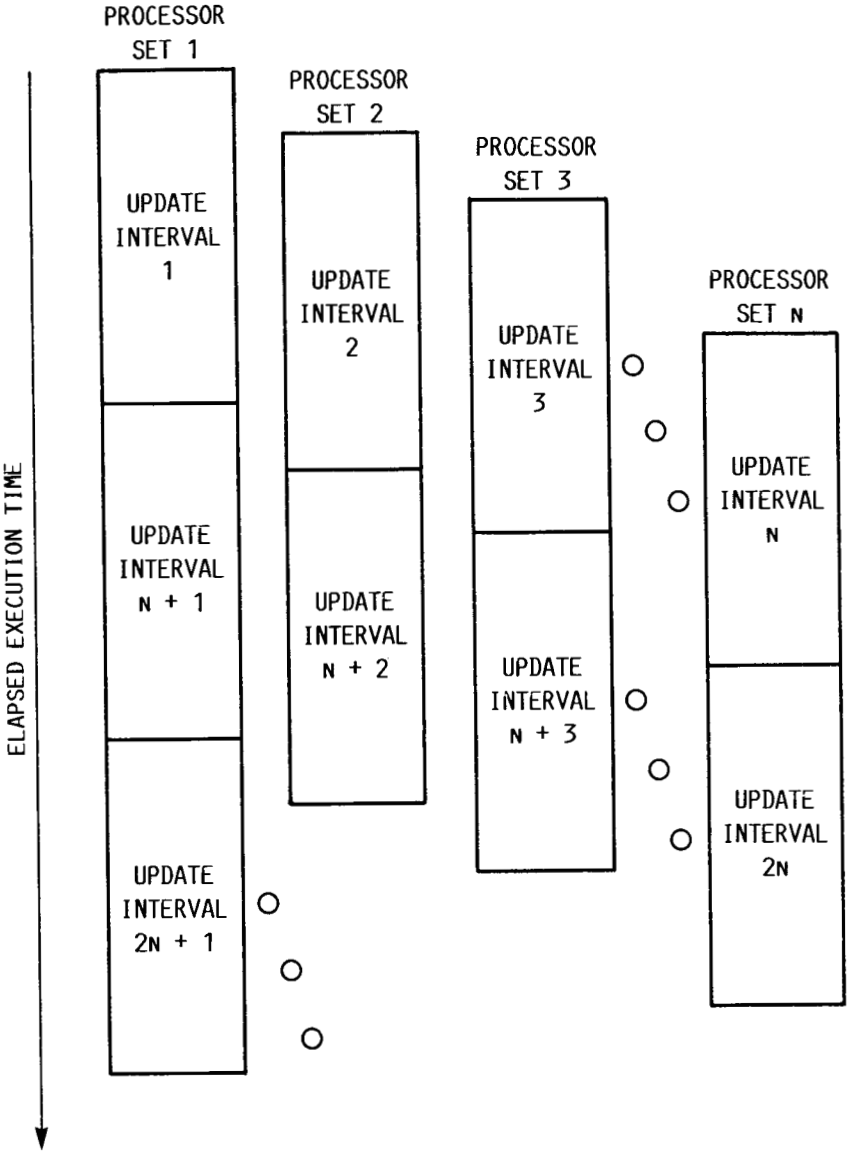

FIGURE 1. - TIME-PARTITIONING EXECUTION PROCESS.

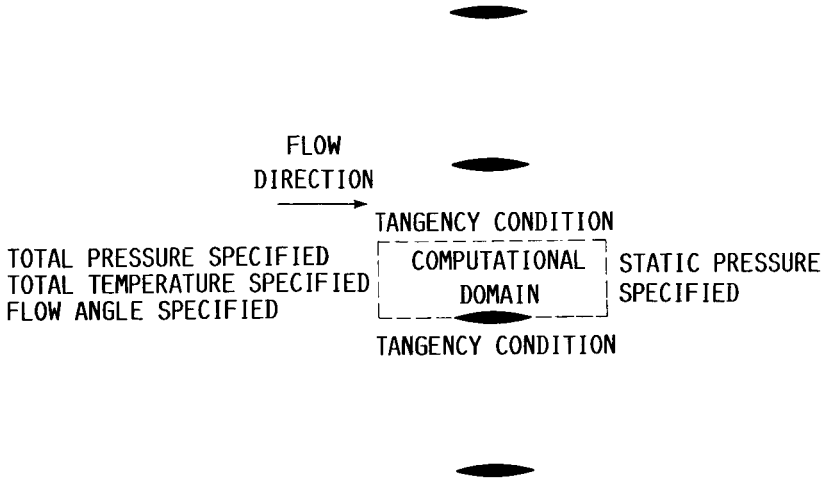

FIGURE 2. - CASCADE OF BICIRCULAR ARC AIRFOILS.

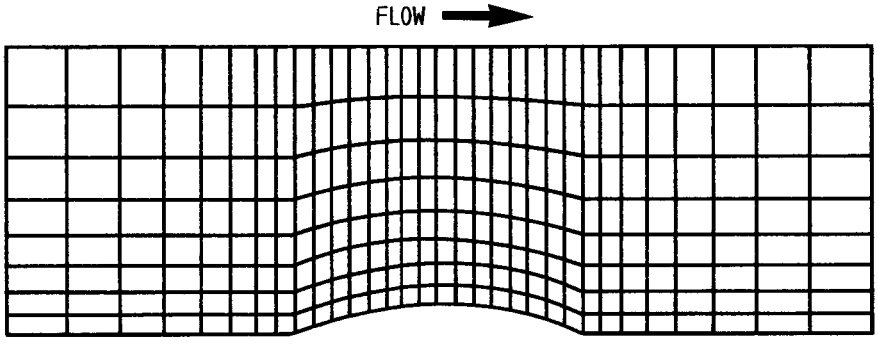

FIGURE 3. - CASCADE OF AIRFOILS SOLUTION GRID.

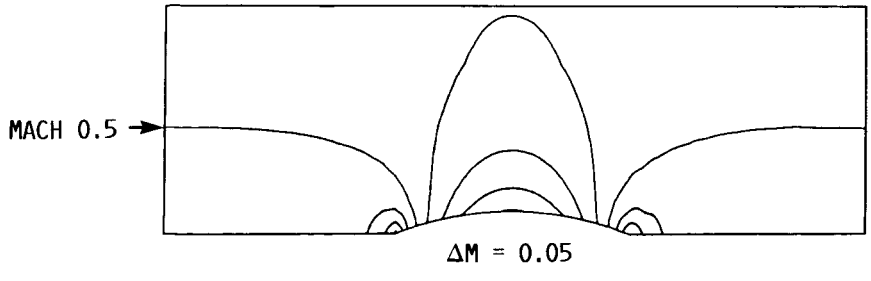

FIGURE 4. - ISOMACHS FOR INVISCID BICIRCULAR ARC CASCADE. MACH 0.5.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-89850		

4. Title and Subtitle	5. Report Date
Time-Partitioning Simulation Models for Calculation on Parallel Computers	6. Performing Organization Code: 505-62-21

7. Author(s)	8. Performing Organization Report No.
Edward J. Milner, Richard A. Blech, and Rodrick V. Chima	E-3517
	10. Work Unit No.

9. Performing Organization Name and Address	11. Contract or Grant No.
National Aeronautics and Space Administration Lewis Research Center Cleveland, Ohio 44135	13. Type of Report and Period Covered: Technical Memorandum

12. Sponsoring Agency Name and Address	14. Sponsoring Agency Code
National Aeronautics and Space Administration Washington, D.C. 20546	

15. Supplementary Notes

Prepared for the 1987 Summer Computer Simulation Conference, sponsored by the Society for Computer Simulation, Montreal, Canada, July 27-30, 1987.

16. Abstract

A technique allowing time-staggered solution of partial differential equations is presented in this report. Using this technique, called time-partitioning, simulation execution speedup is proportional to the number of processors used because all processors operate simultaneously, with each updating the solution grid at a different time point. The technique is limited by neither the number of processors available nor by the dimension of the solution grid. Time-partitioning was used to obtain the flow pattern through a cascade of airfoils, modeled by the Euler partial differential equations. An execution speedup factor of 1.77 was achieved using a two processor Cray X-MP/24 computer.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Time-partitioning Parallel processing	Unclassified - unlimited STAR Category 61

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price*
Unclassified	Unclassified		

*For sale by the National Technical Information Service, Springfield, Virginia 22161