NASA CR 174641
PWA 5896-21

NASA

PARALLEL PROCESSOR ENGINE MODEL PROGRAM

FINAL REPORT

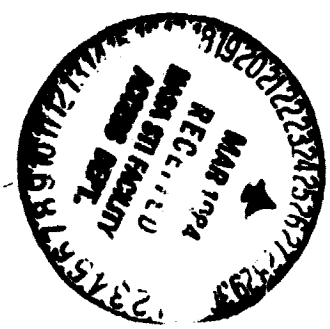
by
Peter McLaughlin

UNITED TECHNOLOGIES CORPORATION
Pratt & Whitney Aircraft Group
Pratt & Whitney Engineering

Prepared for

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
Lewis Research Center

Contract NAS3-23283



1.REPORT NO.	2. GOVERNMENT AGENCY	3.RECIPIENT'S CATALOG NO.
NASA CR-174641	NASA/LEWIS	

4.TITLE AND SUBTITLE	5. REPORT DATE
PARALLEL PROCESSOR ENGINE MODEL PROGRAM FINAL REPORT	January 1984
	6. PERFORMING ORG. CODE
7.AUTHOR(S) Peter Mclaughlin	8. PERFORMING ORG. REPT. NO. PWA 5896-21
9.PERFORMING ORG. NAME AND ADDRESS UNITED TECHNOLOGIES CORPORATION Pratt & Whitney Aircraft Group Pratt & Whitney Engineering	10. WORK UNIT NO. 11. CONTRACT OR GRANT NO. NAS3-23283
12. SPONSORING AGENCY NAME AND ADDRESS National Aeronautics and Space Administration Lewis Research Center 21000 Brookpark Road, Cleveland, Ohio 44135	13. TYPE REPT./PERIOD COVERED Contractor Report 14. SPONSORING AGENCY CODE

15. SUPPLEMENTARY NOTES

16. ABSTRACT

The Parallel Processor Engine Model Program is a generalized engineering tool intended to aid in the design of parallel processing real-time simulations of turbofan engines. It is written in the FORTRAN programming language and executes as a subset of the SOAPP simulation system. Input/output and execution control are provided by SOAPP; however, the analysis, emulation and simulation functions are completely self-contained.

The program had two major objectives. These were to provide a framework in which a wide variety of parallel processing architectures could be evaluated and to provide tools with which the parallel implementation of a real-time simulation technique could be assessed.

17. KEY WORDS (SUGGESTED BY AUTHOR(S))	18. DISTRIBUTION STATEMENT
Parallel Processor Simultaneous Techniques Real-time Non-Linear Turbofan Engine	

19. SECURITY CLASS THIS (REPT)	20. SECURITY CLASS THIS (PAGE)	21. NO. PGS	22. PRICE *

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

FOREWORD

The Final Report contained in this document was prepared under contract NAS3-23283, "Parallel Processor Engine Model Program". The program is intended to aid in the design of parallel processing real-time simulations of turbofan engines. The program was conducted under the direction of Mr. Carl J. Daniele who served as the NASA Program Manager, and Mr. Peter W. McLaughlin who performed as Program Manager at United Technologies Corporation.

TABLE OF CONTENTS

LIST OF FIGURES

I. SUMMARY

Parallel Processing Engine Model (PPEM) is a 12-month program conducted by Pratt & Whitney Aircraft in order to develop a methodology that can be used by NASA in designing real-time digital simulations of turbofan engines. This methodology is embodied in a computer program which is supporting the design of the Real-Time Multi-Processing System (RTMPS), a parallel processing system now being developed by NASA-Lewis Research Center.

The program consisted of the design, development, and submittal to NASA of algorithms by which mathematical models of turbofan engines can be effectively implemented on the RTMPS. A real-time simulation technique (reference 1), based on the work of C. W. Gear (reference 5) and C. G. Broyden (reference 6) and developed at Pratt & Whitney, was utilized in the program. Using this methodology, the implementation of a typical turbofan model on the RTMPS (reference 2) as well as other parallel processor designs was simulated.

The simulation took the form of a serial computer program which was delivered to NASA. This program implemented a parallel version of the real-time technique, model segmentation and distribution algorithms, timing, and storage estimators, and provides the capability to execute test cases using an emulation of the target parallel processing system. This program was used to generate data describing the effect on the technique's accuracy of variations in the parallel processing system's design parameters.

The results obtained in this program show that the use of low-cost microprocessors operating in parallel can achieve real-time turbofan engine simulation capability.

II. INTRODUCTION

Real-time simulation of gas turbine engines has been shown to be an effective and efficient design and development tool. It contributes to productivity in several ways. It reduces the full-scale engine testing required to develop the control system. By replacing very expensive full-scale testing with less costly bench tests, the savings in both cost and time are considerable. Furthermore, the use of real-time simulation can enhance the value of full-scale testing by directing it at problem areas identified by the bench test. System modifications can be investigated prior to hardware commitment. Finally, real-time simulation has proven to be a significant aid in the certification testing of control computer software. Much of this process is accomplished using bench tests employing real-time simulations.

The use of nonlinear, digital simulation in real-time applications, however, has been limited. To address this limitation, Pratt & Whitney has developed the methodology and associated computer programs and documentation for producing a segmented turbofan engine model that can run in real time on a parallel processing system.

The program was based on the use of our previously demonstrated nonlinear real-time simulation technique, which is well suited for this application. This technique, which has been successfully implemented on conventional serial processors, provides numerically stable results even when applied to systems possessing a wide range of time constants (stiff systems). This technique can be applied directly to any continuous nonlinear model without extensive simplification or analytical derivations.

In the development of turbine engine control systems, the use of real-time simulations of the engine has become commonplace. While nonlinear, digital simulation is the standard approach used in other facets of the design process, its use in real-time applications has been limited by two factors:

(1) The need for extremely fast digital processors which can execute the nonlinear model at a rate consistent with the accurate representation of the frequency response characteristics of the engine, and

(2) The availability of techniques which combine absolute numerical stability with the explicit requirements of operation in real time; i.e., a fixed number of calculations in each time step.

In many cases, the accurate nonlinear model is linearized and implemented as a set of transfer functions with variable time constants. These programs are capable of stable, real-time operation on relatively low-cost computing equipment but require considerable development time in order to achieve acceptable accuracy. The use of explicit numerical integration methods, such as Euler or Runge-Kutta, is inappropriate in this application due to the range of time constants associated with turbofan engine models. These methods require the use of update intervals smaller than is necessary for an accuracy adequate in control studies. Generally, this can not be achieved in a cost-effective manner without compromising the validity of the model by over-simplifying it.

In recent years, real-time, nonlinear models of turbofan engines have been successfully executed at Pratt & Whitney on commercially available mini-computers. The technique used in their implementation is based on one utilized in conventional nonlinear transient simulation applications where real-time operation is not required. Here, the use of Gear's backward-difference numerical integration formula requires an iterative evaluation. A quasi-Newton method of Broyden is employed where the elements of the inverted Jacobian matrix are updated on the basis of information obtained from previous iterations. The key feature of this technique is that the Jacobian used need not be re-evaluated during the transient. This results in a highly efficient simulation whose execution rate is relatively insensitive to the size of the update interval. However, because of variations in the number of iterations required to obtain a fully converged solution at each time point, it is unsuitable for real-time applications.

If the number of iterations is held to some arbitrary value without regard to the sizes of the residual iteration errors, then real-time operation is possible. Of course, by relaxing the convergence requirement, the possibility for numerical instability is introduced. Experiments have shown that this phenomena occurs at an update interval that is larger than that required for dynamic accuracy in turbofan engine simulations. It has, therefore, been successfully executed in this application. In fact, with a modification to the Broyden update algorithm, the technique is successful even when only one model execution per update interval is employed.

Wher the real-time technique is viewed as a predictor-corrector method, the corrector step is replaced by an iteration update. This, in turn, is computed from the approximate inverted Jacobian modified by the previous Broyden updates. In this way, the second model execution required in predictor-corrector methods is replaced by a linear solution which corrects the iteration variables on the basis of the iteration errors obtained from the predictor step. The iteration errors are generally the difference between the predicted state iteration variables and those calculated from the integration formula. The loss in accuracy relative to the fully-converged solution is proportional to the residual error resulting from the single corrector step (iteration). This, in turn depends on the size of the update interval employed.

In general, the use of the Broyden update algorithm requires that external inputs to the model be held constant during the iterative process. If it is to be applied to the iteration errors from two consecutive update intervals, it is impossible to maintain this condition unless the effect of the external disturbance can be accommodated. The only practical means of accomplishing this is to treat them as iteration variables and then to account for the changes to them within the Broyden algorithm. An increase in execution time results from the addition of iteration variables but this is normally much less than the execution time saved by the elimination of a second model execution.

In order to implement a parallel simulation technique, it is necessary to segment the algorithm and allocate its elements evenly among several parallel processors. The real-time technique discussed above meets this requirement. As will be seen, it is possible to segment the technique in such a way that each processor can operate independently of the others. The coupling required between the segments residing on each of the processors occurs through the use of block data transfers among them. Furthermore, the segment sizes can be adjusted so as to achieve an optimum distribution of the overall computing task among the processors. For these reasons, it is ideally suited for implementation on a parallel processing system; such as the Real-Time Multi-Processor System (RTMPS) being developed by NASA Lewis Research Center. Furthermore, the structure of the Broyden algorithm can also be adapted to parallel processing. This report is intended to explain the means by which these concepts can be most effectively implemented on the target system, as well as providing computational tools by which other arrangements and communication schemes can be evaluated. In this way, the greater frequency response capability required by advanced technology real-time engine simulations can be achieved with cost-effective computing technology.

Recent experience with real-time simulations executed in support of closed-loop bench control system development testing has served to reinforce our belief in this practice. The models used in these tasks are based on a piece-wise linear approach, however, and suffer from accuracy problems unless the models are updated frequently. This is a time-consuming and expensive process. Linear models also tend to be less flexible in modelling the effects of secondary control systems, such as intercompressor bleeds and stator vanes. Both of these problems could be accommodated by the use of nonlinear models.

Parallel processing offers an alternative approach to real-time turbofan engine simulation. Instead of improving update rates by the acquisition of faster serial processors, slower, less expensive processors operate on different parts of the computing task, thereby achieving a reduction in the time each of them consumes. The effective update rate of this architecture is the longest of the individual execution times.

The next generation of real-time turbofan simulations will be required to execute models consistent with the requirements of multi-loop control systems at update rates several times greater than current practice. At the same time, market pressure will force the utilization of the most cost-effective approaches available. It is quite possible that arrays of low-cost micro-processors may offer an economical means of achieving nonlinear real-time simulation capability.

III. TURBOFAN ENGINE MODEL

This program is intended to demonstrate the capability and the characteristics of turbofan engine nonlinear models operating in real-time on parallel processing systems. Therefore, it is important that the turbofan engine model utilized represent, at least, current practice. It should, in fact, be capable of demonstrating the higher frequency response characteristics that will be required in future real-time applications. Models similar to the typical turbofan engine used in this program have been implemented on serial processing equipment and utilized in several military control system research and development applications. Commercial engine programs could use models of similar scope in the development test and certification process. With the addition of dynamic fluid continuity effects, this model represents the probable next step in real-time engine modelling practice.

Modelling Practice

In the conventional modelling practice employed in PPEP, rotating components are described in terms of their gas flow capacity and thermodynamic efficiency. Combustors are described by the change in temperature achieved for a given level of fuel flow. The fan is represented by an overall map and an inner stream warpage function. These component representations, taken together with the equations of fluid dynamics and thermodynamics, serve to describe the transient and steady-state performance of a turbofan engine.

Several modifications were made to the basic model. First, the dynamic effects of unsteady continuity (volume dynamics) have been added to the pressure calculations in each of the major components of the model. Second, metal temperature states of the hot section are now treated in the same manner as the other states (and the control inputs), in that they form part of the iteration matrix.

The performance aspects of the model used in this program are defined in a conventional manner. Component representations consist of combinations of algebraic and tabular data. Pressures, temperatures, and gas flows describe the conditions at the interfaces between components. The model is defined in a completely implicit fashion.

In order to provide the most stringent test of the parallel technique possible, especially with regard to precision, it was decided to include fluid continuity effects. This provided time constants which ranged from .00025 seconds up to 10 seconds for transient heat transfer phenomena. With time increments expected to be on the order of 10 to 30 milliseconds, the stability of the technique is well tested when applied to this model

The model requires 18 iteration variables in its evaluation. Twelve of these are state variables representing rotor speeds, metal temperatures, pressures, and actuator outputs. The remaining six are required by the component performance representations.

The number of pressure states estimated in the Work Plan to be available in the model did not recognize that the particular form of the model dictates the number of valid pressure states. With reference to the engine configuration representation shown in Figure 1, the use of an explicit relationship between fan outer and inner diameter pressure means that only a single "metering" relationship (i.e., pressure ratio versus corrected flow) exists in the fan. In a dynamic sense, there can be only one pressure state associated with it. In fact, there are only six metering sections in the model used here. Since a pressure state can only be defined in those volumes bracketed by metering sections, this means that there can be only one state in the duct stream, bracketed by the fan map and the duct exit nozzle. Since there is no fan inner diameter map, as such, the high compressor map serves as the inlet metering section for the core stream. There are, then, a total of four metering sections in the core stream for a total of only three pressure states. These are identified in Figure 1 as the burner inlet pressure -- PTBN -- and the low turbine inlet pressure and exit pressures -- PTLT and PTTE. PTDU is used as the duct stream pressure state.

Accommodation of Stiff Elements in the Model

The normal form of Gear's integration formula is applied to those states whose time constant is large compared to the time increment. For the stiff states associated with continuity effects, however, the use of an integral form results in ill-conditioned Jacobian matrices. Therefore, a differential form is chosen which, at very large increments (relative to the time constant), essentially negates the dynamic effect and degenerates to a simple iterative relationship. Figure 2 shows a comparison of the step response of Gear's formula with that obtained from the trapezoidal backward difference formula when applied to a stiff, first-order system. This effect is a key factor in the success of the technique, since it minimizes the destablizing influence of the spurious elements introduced by conventional integration formulae employed in stiff models.

Experience with this approach has been quite good. Accurate responses have been obtained with 16-bit precision at time increments varying from 1 to 35 milliseconds. While current applications are well served by real-time simulation update rates of 10 to 20 milliseconds, future models will be required with response capability an order of magnitude greater. Proof of the technique in the higher frequency regime is an important consideration in its evaluation.

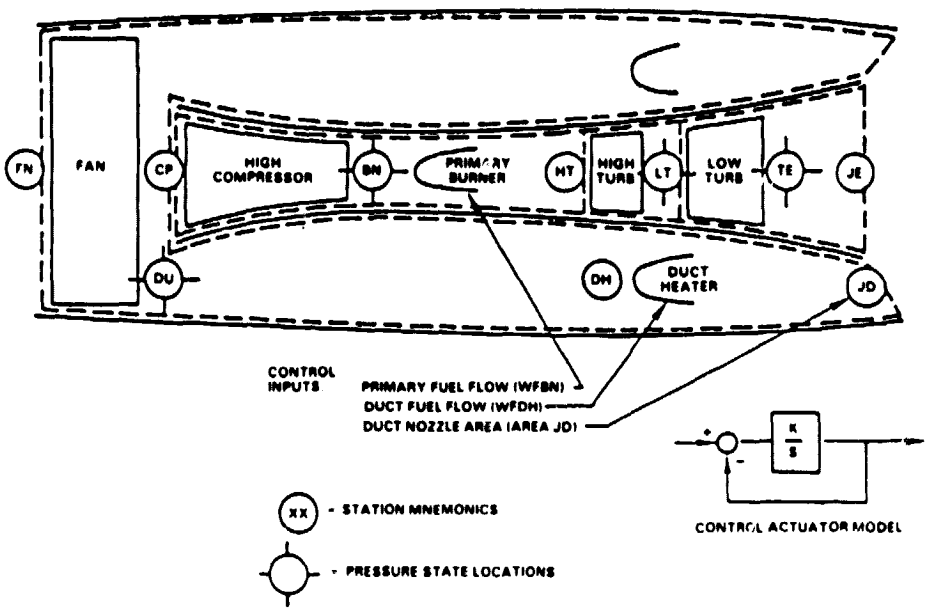


Figure 1 Typical Turbofan Engine Model

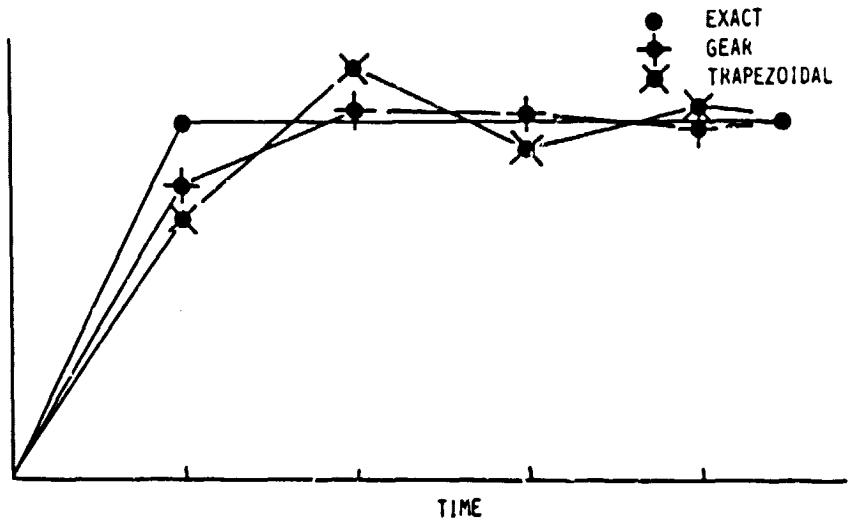


Figure 2 Comparison of the Stiff Response of Backward Difference Integration Formulas

IV. REAL-TIME SIMULATION TECHNIQUE

Broyden-Gear Real-Time Simulation Technique

The Broyden-Gear real-time simulation technique is based on a fully implicit method used routinely in non-real-time applications. The Gear integration formula provides stable response in stiff systems, while the Broyden update algorithm corrects the iteration matrix and provides stable, efficient iterative performance. The truncated version used in real-time applications is capable of stable operation at time increments that are consistent with the capabilities of low-cost computational equipment, while providing adequate response in closed-loop tests with real control systems.

Gear's second order integration formula is employed in this program. This backward-difference integration scheme provides stable operation when applied to stiff (small time constant) states while accurately representing the response of the slower responding elements. The integration formula is given by:

$$X_n = (4 \cdot X_{n-1} - X_{n-2} + 2 \cdot \dot{X}_n \cdot DT)/3 \qquad (1)$$

where:

X - system state

$\dot{X}$ - derivative of the state with respect to time - dX/dt

DT - update interval

Subscripts - denote values at current and past update intervals

When applied to the stiff elements of the system (e.g., pressure dynamics), it is appropriate to employ a differential form:

$$\dot{X}_n = (3 \cdot X_n - 4 \cdot X_{n-1} + X_{n-2})/(2 \cdot DT) \qquad (2)$$

In either case, the state is taken to be an iteration variable, and an error term is formed which is either a function of the calculated state variable or its time derivative.

Gear's second-order formula can be readily expressed in a single-step form, as shown below. This has the effect of simplifying the design of the evaluation facility. If an average derivative is defined as:

$$\dot{X}_{av_n} = [X_n - X_{n-1}]/DT \qquad (3)$$

then, in terms of this variable's past value, the integration formula can be expressed as:

$$X_n = X_{n-1} + [2\dot{X}_n + \dot{X}_{av_{n-1}}]DT/3 \qquad (4)$$

The error term associated with a given state variable is:

$$f(X) = X_n^i - X_n$$

where: X_n^i = iteration variable

In order to aid the iterative process in its evaluation of stiff systems of equations, the differential form of Gear's formula is employed when DT is larger than some characteristic time representative of the state's response rates. If the integral form is used under this condition, the Jacobian elements increase with DT, which eventually leads to near-singularity and a failure to achieve convergence. The value of DT at which the two forms produce equivalent Jacobian elements is given by:

$$DT_c = \frac{3/2}{\dot{\Delta X}/\Delta X} \qquad (5)$$

$1/(\dot{\Delta X}/\Delta X)$ is an equivalent state time constant calculated directly from the model. The second-order Gear differentiation formula is given by:

$$\dot{X}_n = 3/2 \cdot \left[X_n - X_{n-1}\right]/DT - X_{av_{n-1}}/2 = 3/2\ \dot{X}_{av_n} - 1/2\ \dot{X}_{av_{n-1}} \qquad (6)$$

The error term associated with the differential form is proportional to the difference between the derivative computed by the model and that obtained from Equation 6. In order to rationalize the units of the differential error with those of the integral error, it is multiplied by the equivalent state time constant:

$$f(X) = \left[\dot{X}_n^i - \dot{X}_n\right] \cdot 1/(\dot{\Delta X}/\Delta X)\ ;\quad \dot{X}_n^i = f(X_n^i) \qquad (7)$$

In steady-state, an initial value specification replaces the value obtained from the differentiation formula. The net effect of this procedure is that the elements of the Jacobian vary by only a factor of 3/2 for all values of DT.

Convergence of the error terms to within a given tolerance will be obtained by the use of Broyden's quasi-Newton method. The successive values of the iteration variables in a Newton method are given by:

$$X^i = X^{i-1} - f(X^{i-1}) \cdot A^{-1} \qquad (8)$$

where:

$f(X)$ - iteration error term

A^{-1} - inverted Jacobian matrix

superscripts - denote values at successive iteration steps

At each step of the iteration, the elements of the inverted Jacobian are updated by Broyden's method:

$$A^{-1} = A^{-1} + \Delta A^{-1}$$

$$A^{-1} = \frac{\Delta X^T \cdot (\Delta X^i - \Delta f^i \cdot A^{-1})}{(\Delta X^T \cdot \Delta f^i)} \tag{9}$$

where:

$$\Delta X^T = \Delta X^i \cdot A^{-1^T}$$

$$\Delta X^i = X^i - X^{i-1}$$

$$\Delta f^i = f(X^i) - f(X^{i-1})$$

In the real-time version of this technique, only one iteration update is allowed, preceded by the matrix update which is based on the results obtained from the single execution of the model and the integration algorithm.

In order to reconcile the increments (Δf^i) which are associated with state variables with the requirements in the update algorithm of Equation 9, the changes in the state over the update interval must be added to them.

$$\Delta f^i = f(X^i) - f(X^{i-1}) + \Delta X^i \tag{10}$$

where:

$$\Delta X^i = X^i - X^{i-1}$$

External disturbances to the model such as control inputs must be accommodated in the real-time simulation technique. For each of these, a state variable is defined whose derivative is given by:

$$\dot{X} = (C - X)/\tau \tag{11}$$

where:

C is the control input disturbance

τ is a time constant.

The values of the iteration variables used to evaluate the model in the real-time technique are obtained from a predictor whose arguments are the values obtained from previous iteration updates. A linear predictor is used:

$$X_n^P = (2)X_{n-1} + X_{n-2} \tag{12}$$

Equation 8 can then be written as:

$$X_n = X_n^P - f(X_n^P) \cdot A^{-1} \tag{13}$$

It should be noted that the outputs of the simulation are the predicted state and iteration variables. A drawback of the real-time technique is that each external input to the simulation must be matched by a corresponding iteration variable. This adds to the computational load and delays their effect on the model by a full cycle. For this reason, it is recommended that all outputs from the model used to drive the control system hardware be likewise treated as iteration variables which can take part in the corrector-predictor step. The predicted values used to evaluate the model are also used as the output signals.

Broyden Tolerance

Experience with the technique implemented in 16-bit scaled integer arithmetic in this contract has been consistent with earlier results reported in Reference 1. To repeat the conclusions drawn then, the Broyden algorithm operating on numbers of low precision and models of high discontinuity walks a tightrope between two unstable chasms. If the tolerance applied to the scaler factor ($\Delta X^1 \cdot \Delta f^1$) used to normalize the update is selected too large, information necessary to stabilize the simulation may be ignored, while too small a value produces spurious increments caused by trunication errors. Generally, a value can be found in the range between 0.0001 and 0.00001 (normalized), which allows stable response at small enough time increments. Reliability improves greatly when greater precision is specified.

V. PARALLEL IMPLEMENTATION

The basic parallel architecture that was investigated in this project is shown in Figure 3. Several microprocessors are arranged on a data bus which allows high-speed transfers of data among them. A host processor performs control and utility functions. Data can be transferred synchronously (once-per-cycle) on the bus. While outside the capability of the original specification, provisions were made to investigate the use of asynchronous data transfers directly from one processor to another.

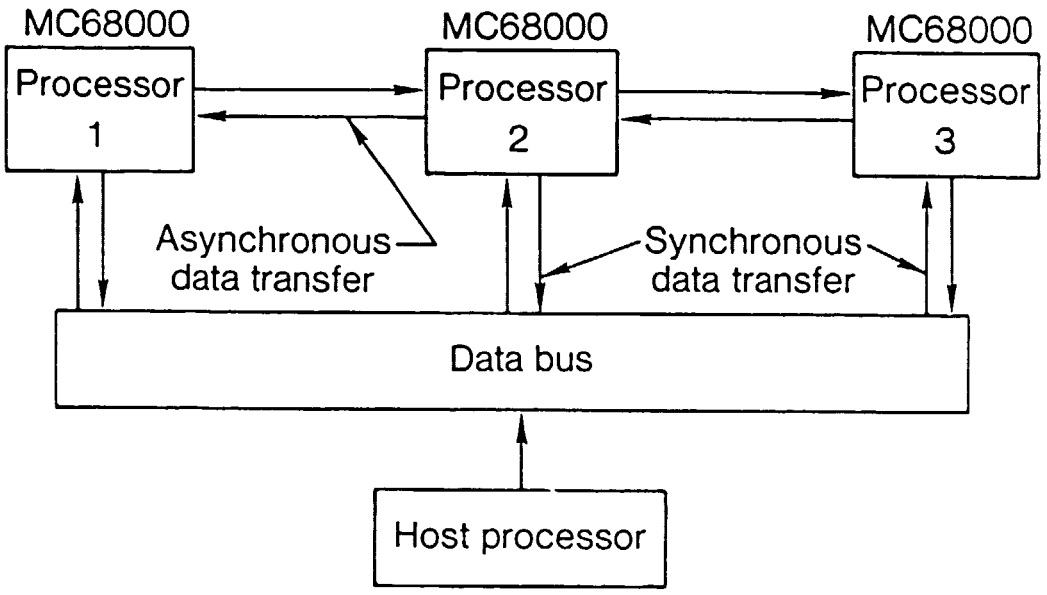


Figure 3 Parallel Processing Architecture

Model Segmentation

The first step toward a parallel implementation of the simulation was to format the turbofan engine model as a series of model segments. The segments are self-contained sections of the component models. They carry with them an identification of the inputs required and outputs computed. This information is utilized in the data flow analysis, which identifies the iteration variables required for a particular parallel implementation.

The overall model was divided into 39 approximately equal segments. In practice, this degree of segmentation was more than adequate to achieve well balanced allocations of the model among the parallel processors.

The segmentation process carries with it an implied order of execution. This order is the one used in its serial implementation. It requires a minimum set of iteration variables for its evaluation in the serial mode. This "preferred" order is used in the allocation process, where individual segments are placed on a particular parallel processor. Rearrangements of the segments required to achieve a given configuration retain this ordering.

Identification of the Iteration Variables

In a given parallel arrangement of the model segments, additional iteration variables are required because the segments on a given processor may require information computed on another processor. In the synchronous data transfer mode, this is the only way to provide the necessary communication path. By creating the additional iteration variables, the model segments allocated in parallel can be coupled in an arbitrary fashion. The algorithm for finding the set of iteration variables required in a particular configuration relies on the data flow information obtained from the segment definitions. Each segment input is examined to see if it has been previously computed on the processor or if it has been previously identified as an iteration variable. If neither condition is met, the segment input variable is added to the list of iteration variables.

Summary of the Technique

A previous section detailed the development of the serial version of the technique. It is a truncated version of an implicit technique used in conventinal non-real-time dynamic simulation. In its real-time manifestation, the technique demonstrates excellent stability on very stiff models; but, since it is an explicit method, it is not A-stable. In the range of update rates experienced in current practice, however, it provides evaluations of correctly defined turbofan engine models with great reliability.

The technique is a quasi-Newton nonlinear solver, where the inverse Jacobian is updated by Broyden's method. Gear's stable second-order integration formula reduces spurious response to negligible levels. The nominal mode for this method is one in which iteration errors are reduced successively until all lie within a given tolerance band--termed fully converged. In the real-time version of the technique, each execution (corresponding to a given time point) is treated as an iteration attempting to converge the errors. With an appropriate modification to the Broyden update, the method reduces to a variety of predictor corrector, where predicted values are used to evaluate the model and are then corrected by the inverted Jacobian obtained from Broyden. The corrected values form the basis for the next prediction and are output to the control system hardware. The Broyden algorithm utilizes changes in both iteration variables and errors, as well as the current inverse Jacobian to compute changes to that matrix required to attain convergence.

Parallelization Considerations

Although the algorithm appears computationally complex, the communication of data between the different elements is quite manageable. This, coupled with the flexibility afforded by an implicit model formulation, makes the technique an attractive candidate for parallelization. In this discussion, it will be assumed that the transfer of data between parallel processors can only occur once per time step.

For a given arrangement of model segments on the available processors, a set of iteration variables necessary to solve the set of distributed equations in a simultaneous fashion can be identified. Each processor must be capable of calculating updated values of the iteration variables required on that processor, and it must compute errors associated with the values of iteration variables calculated in the model. This specifies which elements of the vectors involved are to be computed on a given processor.

Figure 4 shows the inter- and intra-processor data flows required by the real-time technique. This diagram also shows the arrangement of program components on one of the parallel processors. The solid lines lying within the boundaries of the processor indicate data flow between elements on that processor. The double lines show data that must be transferred to other processors and be received from them. In the arrangement shown, the operations performed in Broyden depend on the distribution of iteration variables among the processors.

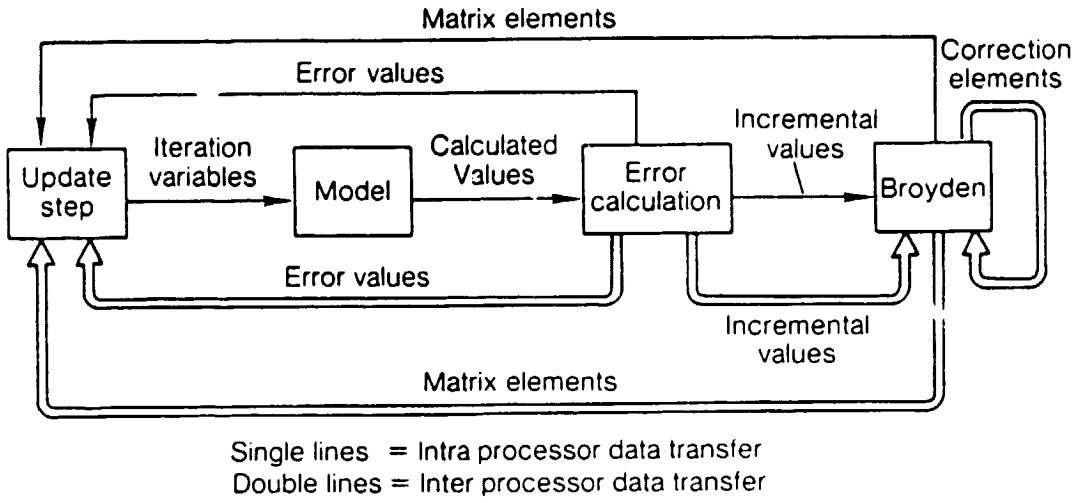


Figure 4 Data Flow Requirements - Coresident Broyden Algorithm

This diagram illustrates the source of a destabilizing effect of the parallel implementation. Data is collected from both the error and Broyden vector calculations but is delayed in its effect on the matrix for a full cycle, as it is exchanged among the processors. Data distributed from the error calculations and used in the iteration variable update is, of course, undelayed.

While this technique provides improved execution rates, it suffers from a major deficiency. As more processors are added and the model is distributed further, greater numbers of iteration variables are required. This leads to timing penalties that overwhelm the improvement derived from parallelization. This appears to be a fundamental limitation of synchronous data transfer applied to this technique.

An alternative approach was devised where the Broyden algorithm could be executed simultaneously with the remainder of the simulation. At least half the processors are used for Broyden, which reduces the number of iteration variables required in a given processor configuration. Since the allocation of Broyden computations is no longer connected to the model arrangement, it can be distributed in such a way as to balance the computing load among the processors. The cost of this approach is an additional full cycle delay in the matrix calculation. The fact that faster update rates are achieved does tend to compensate this effect.

Figure 5 shows the arrangement of two processors where the Broyden algorithm is executed simultaneously with the remainder of the technique. An additional penalty is the need for greater transfer capability.

This arrangement proved to be capable of achieving adequate real-time performance applied to the RTIPS specified by NASA-LeRC; namely, eight to ten MC68000 processors serviced by synchronous data transfer. Data presented later in this report documents the results obtained.

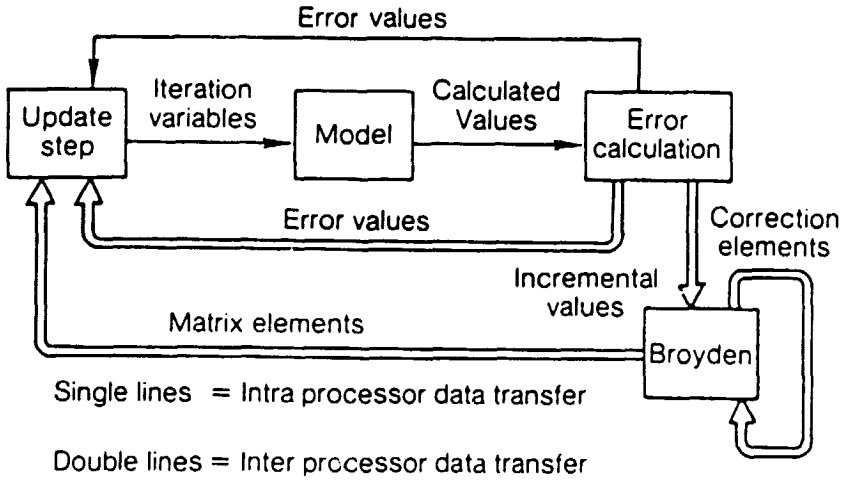


Figure 5 Data Flow Requirements - Simultaneous Broyden Algorithm

Parallel Model Allocation

The basic strategy implemented is to allocate the model segments in some fashion and then identify the iteration variable set required to evaluate the model, based on a set of optional parallelization schemes. Figure 6 shows, in schematic, a typical arrangement of model segments allocated among several parallel processors being controlled by a host computer. This device would also handle the data distribution and program downloading tasks. Figure 7 shows lists of iteration variables associated with two of a number of parallel processors. The iteration variables are required for certain computations in segments allocated to that processor. Other segments compute provisional values of the iteration variables. The error term is the difference between the iteration variable and the provisional value computed by the model.

In the synchronous data transfer mode, the iteration variables required on more than one processor must be computed on each processor from data that is obtained from the other processors. The data transfer requirements represent a significant fraction of the overall execution time required for a given parallel configuration. The overall time increment required is the largest of the individual processor elapsed times plus the time required to transfer data. An underlying consideratin in the choice of the parallelization scheme is a need to minimize the amount of data that must be transferred between processors.

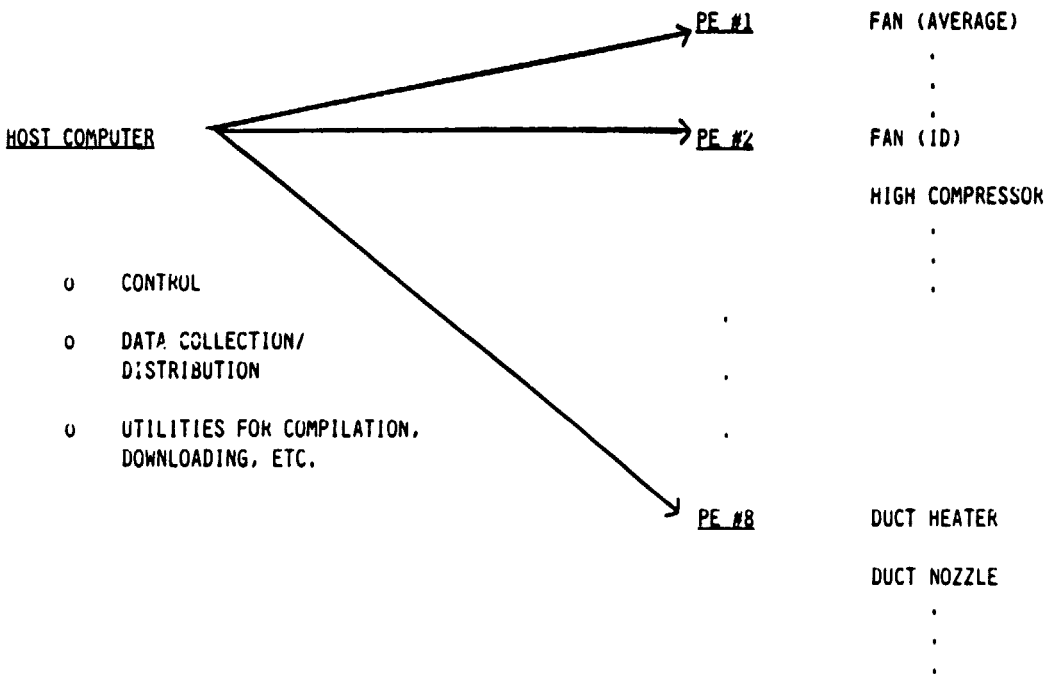


Figure 6 Simulator Functional Structure

PE #1 - FAN (AVERAGE)

ITERATION VARIABLES REQUIRED

RPM1
WCFN
o
o
o

COMPUTED ITERATION VARIABLES

WCFN
o
o
o
o

PE #6 - LOW TURBINE

ITERATION VARIABLES REQUIRED

RPM1
PTLT
o
o
o

COMPUTED ITERATION VARIABLES

RPM1
PTLT
o
o
o

Figure 7 Parallel Implementation of the Technique

Model segments are arranged on the processors in such a way as to minimize the number of iteration variables required. Using synchronous data transfer the number of iteration variables increases rapidly with the number of processors utilized as more variables become unknown on the processor where they are needed. Due to this effect, improvements in the time increment are small and, in fact, disappear as the number of processors increases.

While iteration variable update and error calculations must reside on the processor where the iteration variable is required or calculated, the Broyden matrix update calculations can be placed on separate processors. The reason for this is that the matrix calculation can be based on data that was obtained during earlier iterations. The numerical stability of this simultaneous approach is a key factor in the achievement of real-time capability on the RTMPS configuration.

By allocating model segments to less than half of the available processors, the number of iteration variables is kept to a reasonable level. The remaining processors are used for Broyden and the matrix update. The distribution of this computing task is adjusted so as to attain a balance between it and the remainder of the technique.

Automatic Segment Allocation

This facility is intended to distribute the model among available processors while maintaining a given order of execution. The order of execution is determined by either the default arrangement of the original serial model or an order supplied by the user.

The first step in the allocation process is to find the total execution time required for all segments. This quantity, divided by the number of processors, is taken to be a target processor execution time. In accordance with the user's chosen allocation option, segments are placed on a given processor until the target time increment is exceeded. When this fact is detected, the next processor is utilized, and so on, until all available processors are used. The first attempt will, in general, fail to achieve the original target. That value is increased by a selected fractional amount, and the allocation is repeated until the criteria is satisfied.

Two allocation options are provided. In Option C, the given segment order is used to allocate segments on a given processor, as shown in Figure 6. The directonal indications denote the manner in which segments are allocated using the given ordering data. In Option 1, segments are allocated on successive processor, rather than sequentially, as in Option C.

Parallel Broyden Algorithm

A key element of the parallel real-time simulation technique is the manner in which the Broyden matrix update is distributed among the processors. The Broyden algorithm makes use of several vectors computed from other elements of the quasi-Newton method to compute changes to the inverted Jacobian matrix. The following vectors of incremental quantities begin the Broyden process:

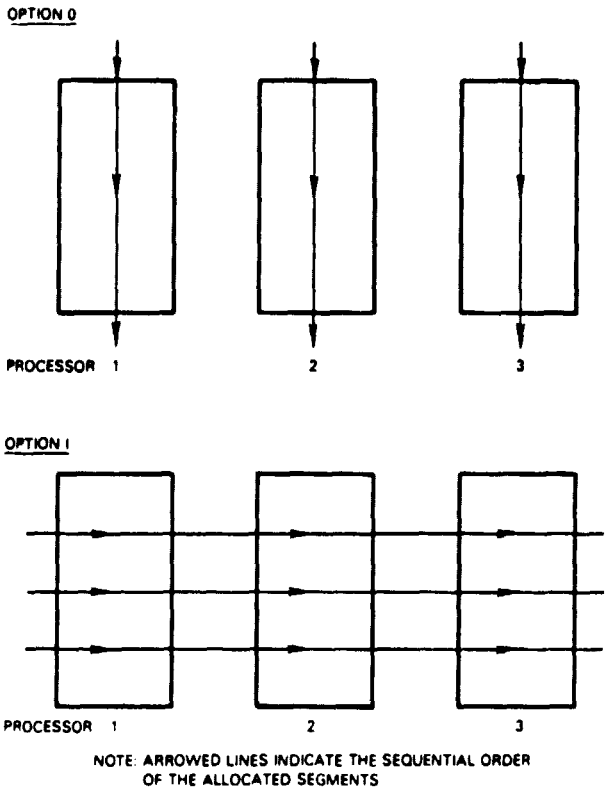


Figure 8 Allocation Options

$$\Delta x^i = x^i - x^{i-i} \tag{14}$$

$$\Delta f^i = f^i - f^{i-i} \tag{15}$$

In the parallel implementation, elements of these vectors are computed on the processors where the iteration variable associated with the element is calculated, and hence, where the error term is computed.

The term "co-resident" is used to describe a parallel scheme in which the Broyden algorithm occupies the same processors as the model and its basic quasi-Newton algorithm. Figure 4 shows this arrangement schematically. A version in which Broyden executes simultaneously with the model, etc., on its own processors will be discussed.

The incremental quantities computed on a given processor are operated on by the current inverted Jacobian matrix to create individual contributions to the Broyden correction vectors:

$$\Delta x_k^1 = \Delta f^i \cdot A^{-1} \tag{16}$$

$$\Delta x_k^T = \Delta x^i \cdot A^{-1^T} \tag{17}$$

where the subscript "k" denotes the contribution of one of the processors. It is made up of individual elements calculated from the incremental quantities available on the processor and the rows or columns of the matrix associated with the iteration variables calculated on the processor.

The matrix update step requires full vectors of each of the four quantities cited above in Equations 14 to 17. Therefore, the individual contributions to the correction vectors (Equations 3 and 4) must be transferred to each of the processors. In the second stage of the process, the full vectors are computed on each processor from an array of individual contributions:

$$\Delta x^1 = \sum_k \Delta x_k^1$$

$$\Delta x^T = \sum_k \Delta x_k^T \tag{18}$$

where "k" denotes that contributions from each of the processors are summed. These two vectors and full vectors of the incremental quantities of Equations 14 and 15 (which also result from data transfer among the processors) are sufficient to compute increments to the matrix elements.

$$\Delta A^{-1} = \frac{\Delta x^T (\Delta x - \Delta x^1)}{(\Delta x^T \cdot \Delta f)} \tag{19}$$

The parallel version of the Broyden update algorithm has produced stable response in both fully converged and real-time modes. Through various program input options, either the co-resident (model and update code sharing a processor) or the simultaneous version can be invoked.

The simultaneous version shown in Figure 5 can be further distributed by splitting it into two parts and executing them on separate processors. This is numerically identical to the basic parallel approach and does not impact stability. Finally, following this line of reasoning further, the Broyden update can be distributed among its own set of processors in such a way as to balance the execution time requirement between the model and update. In this mode, the model execution time becomes the controlling factor. This approach provides the most effective distribution of the overall computing task.

Summary of the Parallel Technique

In the parallel version of the real-time simulation technique, the overall computing task is divided among the processors. These include the four major elements of the process. Figure 4 shows the arrangement of these elements on a single processor. The elements are (1) an iteration variable update, (2) model execution (3) error calculation, and (4) Broyden matrix update.

The allocation of these tasks is based on the requirements associated with the model segments residing in the processor. These are the iteration variables required on a processor and the error terms computed on them.

While iteration variable update and error calculations must reside on the processor where the iteration variable is required or calculated, the Broyden update and matrix calculations can be placed on separate processors as shown in Figure 5. The reason for this is that the matrix calculation can be based on data that was obtained during earlier iterations. The numerical stability of this simultaneous approach is a key factor in the achievement of real-time capability.

By allocating model segments to less than half of the available processors, the number of iteration variables required is reduced. The remaining processors are used for Broyden and the matrix update. The distribution of this computing task is adjusted so as to attain a balance between it and the remainder of the technique.

Data Distribution Requirements

In understanding the parallel implementation of the technique, the data flow requirements among the several components are the most important consideration. Figure 10 is intended to illustrate these requirements. The double lines show data that must be transferred from one processor to another when the parallel version of the technique is implemented.

The technique requires that the errors calculated from a given model execution must be transferred to the other processors prior to their use in updating the iteration variables. If data is transferred synchronously, at the beginning of a cycle only, this fact requires that the update step be executed first, prior to the execution of the model. In the conventional, serial implementation, it takes place at the end of the cycle. Numerically, they are identical. However, data output from the simulation is delayed by a full cycle in the parallel implementation, a fact that should be compensated for by providing an additional stage of prediction for those variables that are to be output.

The next data transfer requirement of the parallel implementation is associated with the Broyden update. Two vectors must be generated from the two change vectors computed in ERROR and the inverse Jacobian matrix. Each processor computes a contribution to each of the elements of the Broyden vectors. These vectors are distributed among the processors where, on the next cycle, they are summed to produce the required vector elements. The updates for those matrix elements required on the given processor are then computed.

The final data transfer path identified is also a requirement of the matrix update. The two change vectors are required in the matrix update step; but, instead of only those elements computed on the processor and used in the Broyden vector computations, the matrix update requires that the entire change vectors be available. These are transferred among processors in a manner similar to that required for the errors.

The parallel Broyden algorithm requires that not only the rows of the inverse Jacobian matrix, which are necessary to update the iteration variables, but the columns, as well, which are necessary for the transpose operation, be available on a given processor. A choice exists in the manner by which the column data is obtained. It can be computed directly on the processor where it is required, or it can be obtained from the row data computed on other processors. These two options are termed resident and distributed, respectively.

Data Transfer

Figure 9 illustrates the two data transfer techniques that were investigated in the contract. Synchronous data transfer was used to distribute vectors and matrices of data among the processors once per model update; that is, once for each time point computed. In addition, the number of iteration variables required is reduced by transferring model variables computed on one processor to another in an asynchronous manner.

Figure 10 shows the data transfer paths required for the parallel implementation of the real-time simulation technique. This section describes the algorithms by which the times required to perform data transfer operations are computed.

The first case that will be analyzed is a vector whose elements are distributed among the processors. It is necessary to make the entire vector available on all the processors. The vector elements on each processor must be transferred to each of the other processors. The total transfer requirement is given by:

$$T = \sum_{NPROC} VECTOR_{PROC} * (NPROC - 1) = NITV * (NPROC - 1)$$

where:

$VECTOR_{PROC}$ - number of vector elements on a processor

NPROC - number of processors

NITV - length of the vectors = $\Sigma_{NPROC}\ VECTOR_{PROC}$

The amount is reduced by the fact that part of the overall vector is already resident on the processor where it is computed.

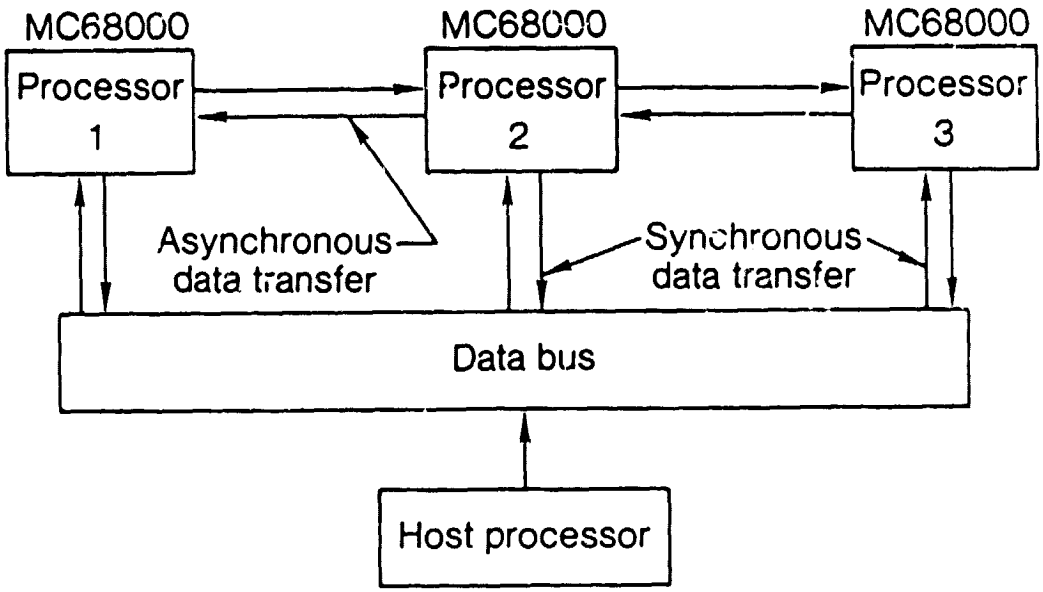


Figure 9 Parallel Processing Architecture

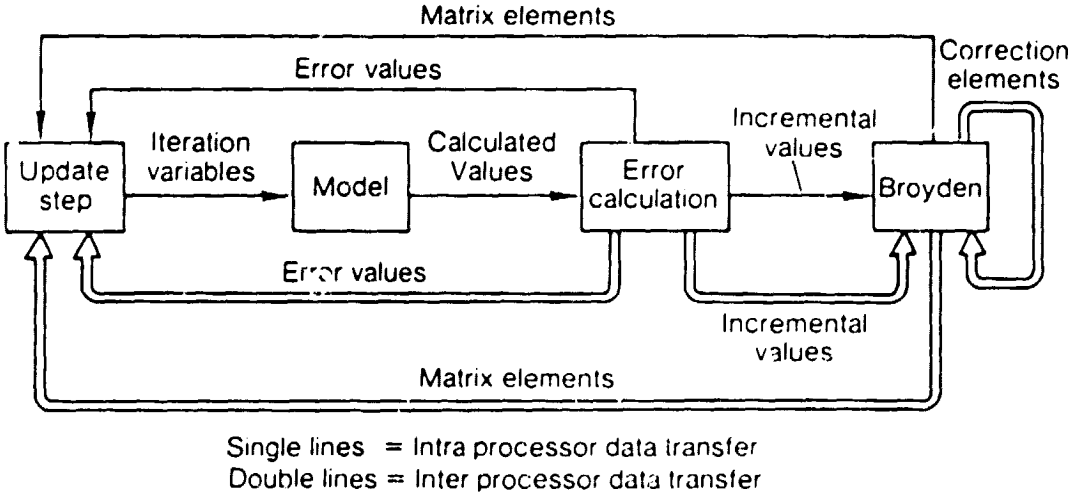


Figure 10 Data Flow Requirements - Coresident Broyden Algorithm

This formula is used to compute data transfer requirements for the error vector and the incremental vectors required by the Broyden algorithm. The total is then:

$$N = 3 * NITV * (NPROC - 1)$$

Transfer requirements for the Broyden correction vectors computed from the incremental vectors are complicated by the fact that pieces of each vector element are computed on each processor. Using similar reasoning, the Broyden correction vector distribution time is given by:

$$N = 2\sum_{NPROC} VECTOR_{PROC} * NITV * (NPROC - 1)$$

$$= 2 * NITV^2 * (NPROC - 1)$$

The factor of two arises from the fact that there are two correction vectors that must be transferred. If the matrix elements required for the iteration variable update and the Broyden correction vector calculations are calculated on the processor where they are used, they require no distribution. If the option is chosen to only compute the matrix elements that correspond to the error terms, then Figure 10 shows that the entire matrix must be transferred, less those elements that already are available on a given processor. Data that must be transferred to more than one other processor must also be included. This requirement is given by:

$$N = \sum_{NPROC} (NITV - NITVC) * NITVR$$

where:

NITV = total number of iteration variables

NITVC = number of iteration variables computed on the processor

NITVR = number of iteration variables required on the processors.

When the simultaneous form of the Broyden algorithm as shown in Figure 11 is requested by the user, the data distribution requirements are modified. In particular, the two incremental vectors now must be distributed to all the Broyden processors. These operations require:

$$N = 2 * NITV * NPROCB$$

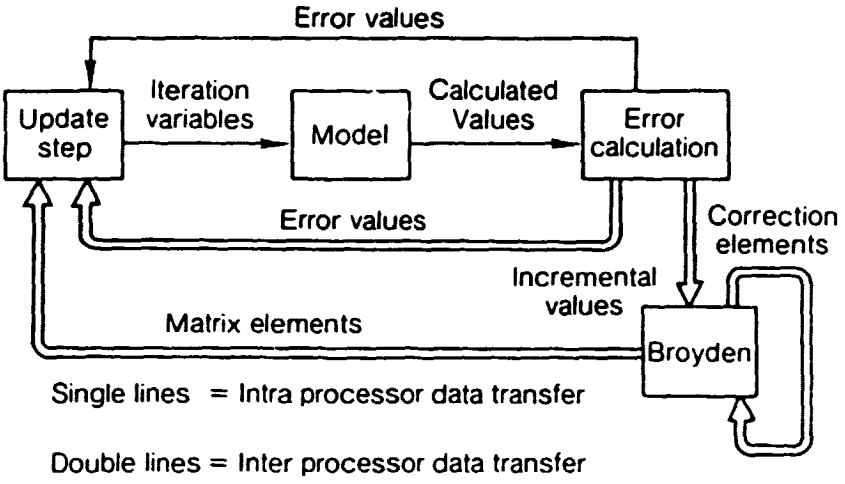


Figure 11 Data Flow Requirements - Simultaneous Broyden Algorithm

where:

NPROCB = number of processors used for Broyden.

Distribution of the error vector still requires:

$$N = NITV * (NPROC - 1).$$

Matrix distribution to those processors where the iteration variables are updated amounts to transferring:

$$N = NITV * NITVR$$

where: NITVR = total number of iteration variables required items.

In the simultaneous mode, matrix elements must also be distributed to the other Broyden processors. This count is given by:

$$N = \sum_{NPROC} (NITV - NITVC) * NITVC$$

Finally, when the Broyden correction step is run simultaneously with the matrix update step, the matrix must be completely distributed, both rows and columns from the matrix updating processors to those computing the correction vectors.

These operations are counted by the following summation:

$$N = \sum_{NPROC} 2 * NITV * NITVC - NITVC^2$$

Asynchronous Data Transfer

An asynchronous data transfer approach has resulted in substantial execution time improvements over the synchronous scheme originally treated. This method utilizes "crosstalk" between the processors to transfer data resulting from model calculations on one processor to another where that data is required for other calculations. This approach represents an alternative to the use of iteration variables as a means of communicating this data.

In this scheme, only the basic set of iteration variables required for the sequential model is allocated. It is assumed that the iteration variable update step will be synchronized so that all the model executions begin simultaneously. This ensures that the results of the iteration variable update will be available for "crosstalking" to processors other than the one where the update takes place.

Figure 12 shows the schematic arrangement of segments in an asynchronous crosstalk scheme. The segment on the receiving processor must be delayed by an amount which is the difference between the ending time of the sending segment and the beginning time of the receiving segment.

In order to prevent contention between processors for precedence in the crosstalk hierarchy, the processors have been ranked in ascending order of their identification number. Crosstalk is only allowed to occur from lower-numbered to higher-numbered processors. Since this is the direction in which the default segment sequences are allocated, there is no requirement for crosstalk in the opposite direction from that preferred. For other segment arrangements, variables that must be crosstalked "upstream" are identified as iteration variables, rather than crosstalk variables, and are treated in the same way as the basic set of iteration variables associated with the sequential model.

The remainder of the data transfer requirements in this arrangement are accomplished synchronously. The only difference between this scheme and the purely synchronous methods discussed earlier is the elimination of iteration variables as a means of communicating data between model segments. Instead, the values of the iteration variables are transmitted asynchronously to the processors requiring them.

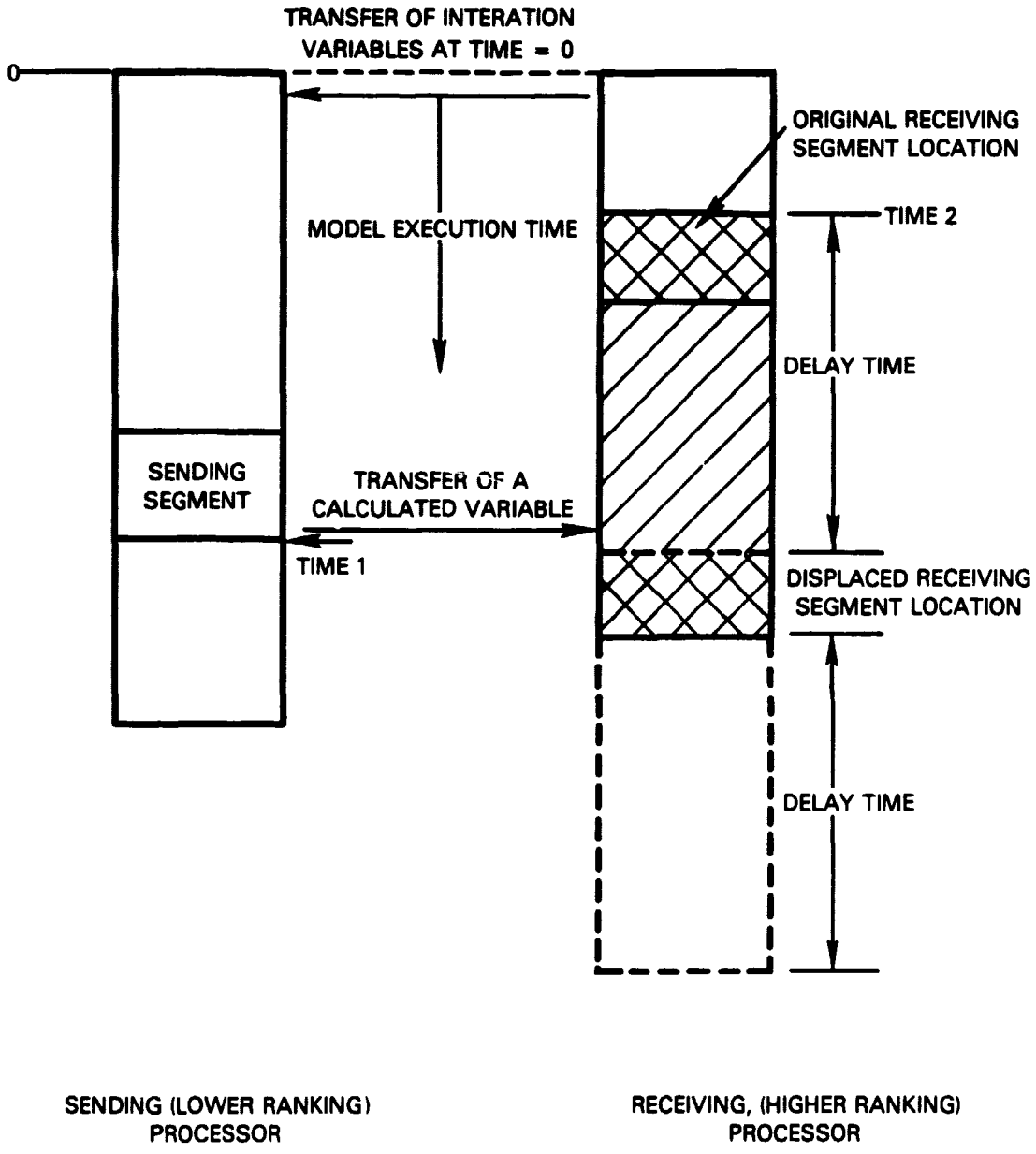


Figure 12 Illustration of Asynchronous Data Transfer Crosstalk

VI. COMPUTER PROGRAM

The methodology developed in this contract has been implemented in a digital computer program. It is intended that the program be utilized in the design and programming of the RTMPS. The two primary tasks performed by the program are:

(1) Validation of the real-time simulation technique in its parallel implementation.

(2) Determination of real-time performance of various parallelization approaches.

The program consists of two major elements. These are the model (including the real-time simulation technique) and the emulation of a generalized parallel processing system. It is written in the FORTRAN programming language and executes on conventional serial processors.

Program Structure and Operation

The Parallel Processor Engine Model (PPEM) program executes as a self-contained subset of the SOAPP simulation system.

Input/output and program control functions are provided by SOAPP system utilities. In addition, the model can be executed using the SOAPP version of the evaluation technique.

The turbofan engine subroutine employs low-level functions to perform the basic arithmetic and program control functions. In these routines, timing storate data is collected and stored for use in parallelization routines. Data flow information is also provided in the engine subroutine, which describes the inputs and outputs of the model segments. In response to user-supplied parallel processor specifications, the program designs the arrangement of the model and the evaluation routines that implement the real-time technique. It then executes the simulation as it would run on the target system.

Multiple execution capability is provided. Each task is completely independent of those that precede it. Input parameters retain previously specified values unless altered by the user. A task may consist of several individual cases. SOAPP provides automatic parameter cycling through input specifications. This capability is used to execute arrays of parametric variations, as well as transient cases.

A case consists of an allocation and configuration step, a fully converged steady-state point, and, optionally, a transient. The first two steps are executed only on the first point of a case. Subsequent executions on a given case compute successive transient points until the user-specififd elapsed time has expired.

Timing data used in this program was taken from reference 7. Data bus specifications were received from NASA-LeRC. Generally, an eight-megahertz processor clock rate and a data transfer rate of one megahertz were assumed.

The timing algorithms are based on reducing the four model's equations down to the level of assembler code. Each instruction is represented as a reference to a low-level FORTRAN function. In these functions, both timing and storage increments are added to running totals and stored for later use. Data transfers between registers and memory are bookkept at a higher level.

Comparisons with known benchmarks reveal that a 15-percent increase in the time increment can be expected in a typical "real" implementation when this timing method is utilized.

The computer program and its operation are fully documented in the Parallel Processor Engine Model Program User's Manual, which is delivered as a companion volume to this Final Report.

Additional Capability

Several optional features were included in the program developed in this effort whose effect on the results obtained was not determined. The most significant of these is the capability to specify the size of a minimum matrix element that will be utilized in the computations. Since much of the iteration matrix tends to be insignificant, it is intended that the much smaller time it takes to make the test, compared to the multiply operation that is disabled, will result in an appreciable reduction in the time increment required for the matrix computations. This is a major element of the overall requirement.

While the use of scaled fixed-point arithmetic provides a kind of scaling effect on the error terms, the use of a separate normalizer applied to the error might result in better performance of the technique. Two forms of normalization have been provided. The first divides the error difference by the iteration variable itself to form a nondimensional quantity. The second method provided uses the largest matrix element of the row associated with the error; that is, the largest partial derivative, as the normalization divisor. Both schemes tend to rationalize the matrix so that the process of scaling it can be performed with more precision. This, in turn, may improve the precision of the update and reduce the size of the residual errors. However, this is achieved at the cost of an increase in the time increment requirement.

VII. RESULTS

In order to demonstrate the capabilities of the algorithms implemented, a series of program tests were executed. In general, these were broken down into two groups. First, the real-time execution rates for a wide range of parallel configurations were computed by the program. From this data, a target configuration was chosen which was used as the vehicle for the second group of tests. These were concerned with the accuracy and stability of the technique as implemented on this configuration.

Configuration Survey

It was recognized early in this work that the use of a detailed emulation of the parallel processing systems under consideration would be invaluable in examining alternative approaches. This FORTRAN program, executing serially, allows the user to evaluate a wide range of parallel arrangements.

Based on the timing data collected from the model and the evaluation routines, and the arrangement of simulation functions chosen by the user, a time increment requirement is computed as the sum of the largest execution time of the processors plus the time required to transfer data among the processors. A large matrix of configurations was executed using the NASA-LeRC Real-Time Multi-Processor Simulator as the target. In these runs, the model was distributed among one to eight processors. Four variations of the parallel technique were executed in both the synchronous and the asynchronous data transfer modes. This resulted in a total of 64 different configurations being evaluated. The synchronous data transfer mode which produced the minimum time increment, and which lies within the design parameters of the RTMPS, utilizes five processors. The model is distributed between two processors. The simultaneous Broyden function is implemented on the remaining three processors. The time increment for this configuration is .0244 seconds. This represents a reduction factor of 54 percent, based on the one-processor requirement of .0536 seconds. Figure 13 tabulates the results obtained from this study. The co-resident Broyden algorithm produces reduction factors consistently inferior to the simultaneous version. As discussed earlier, the co-resident approach distributes the model among more processors and, hence, requires more iteration variables.

Figure 13 also shows that the differences between the resident and distributed matrix options are relatively small. This tradeoff is strongly dependent on the relative magnitudes of the processor clock rate and the data transfer rate. Therefore, hardware with characteristics other than those used in this study might yield different results.

As can be seen from these results, the use of asynchronous data transfer, in some form, offers a significant improvement in the update rate possible with this technique. A miminum value of .0159 seconds was obtained for an asynchronous configuration consisting of nine processors.

Figures 14a through 14f show typical results obtained in this study. The first three show the overall time increment required, the simulation execution time, and the data distribution time for the simultanous Broyden, distributed matrix update, synchronous data transfer configuration as functions of the number of processors utilized. The next three figures show this data for the simultaneous Broyden, resident matrix update, asynchronous data transfer configuration. The most important characteristic of this data is the comparison between the trends for the two data transfer modes. The ability of the asynchronous mode to provide a reduction in the overall execution time is the most significant finding of this study. Not only is the execution time increment reduced as more processors are added, but, in addition, the data distribution time increment is less than the synchronous requirement by an order of magnitude for a like number of processors.

The configuration employing synchronous data transfer and six processors, and requiring a time increment of .0252, was chosen as a base for further investigations. The basic characteristics of the parallel version of the real-time technique were determined through the use of typical transient disturbance tests. In addition to the engine parameters chosen to illustrate the respone, the root-mean-squared error, computed from each of the individual error terms, is used to give a quantitative estimate of the relative accuracy of the technique.

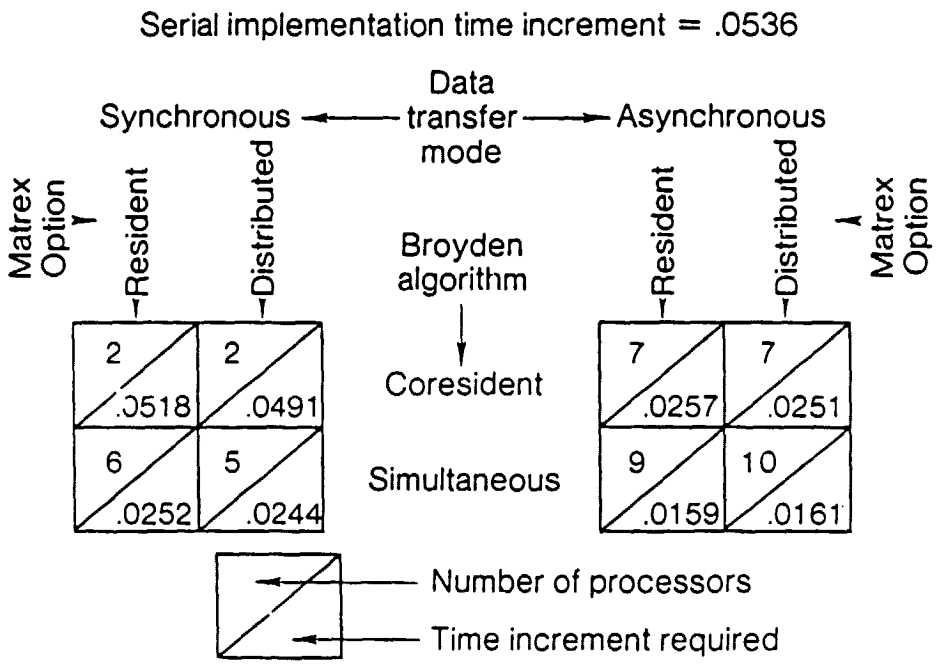


Figure 13 Configuration Survey Summary

PPEM CONFIGURATION SURVEY
SYNCRONOUS DATA TRANSFER
SIMULTANEOUS BROYDEN ALGORITHM
DISTRIBUTED MATRIX OPTION

4 ▲ ■■■ PARALLEL PROCESSOR ENGINE MODEL ■■■ 6677729VU1 (ST 1:

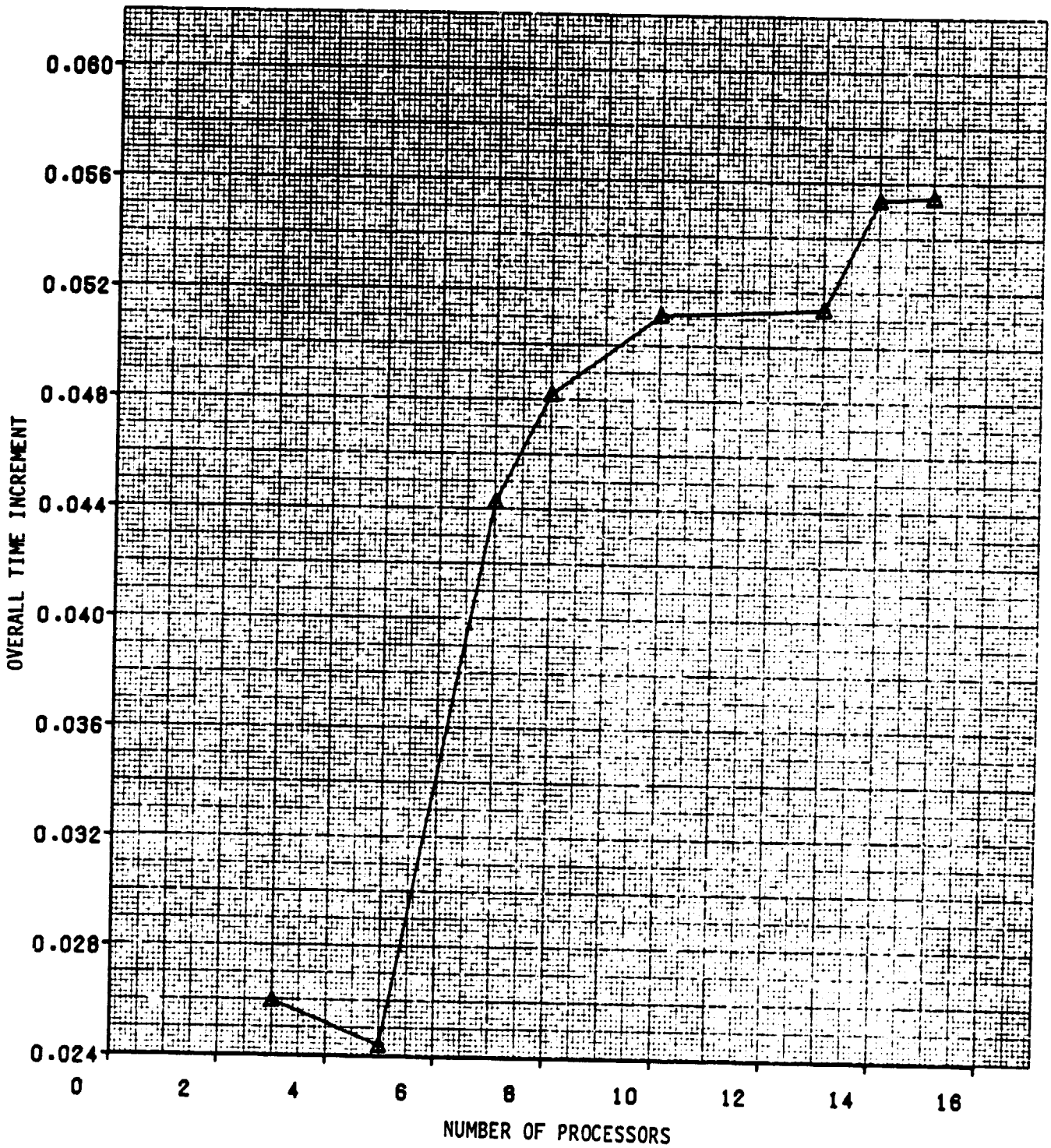


Figure 14(a) PPEM Configuration Summary

PPEM CONFIGURATION SURVEY
SYNCRONOUS DATA TRANSFER
SIMULTANEOUS BROYDEN ALGORITHM
DISTRIBUTED MATRIX OPTION

4 ▲ ■■■ PARALLEL PROCESSOR ENGINE MODEL ■■■ 8677729VU1 (ST 1)

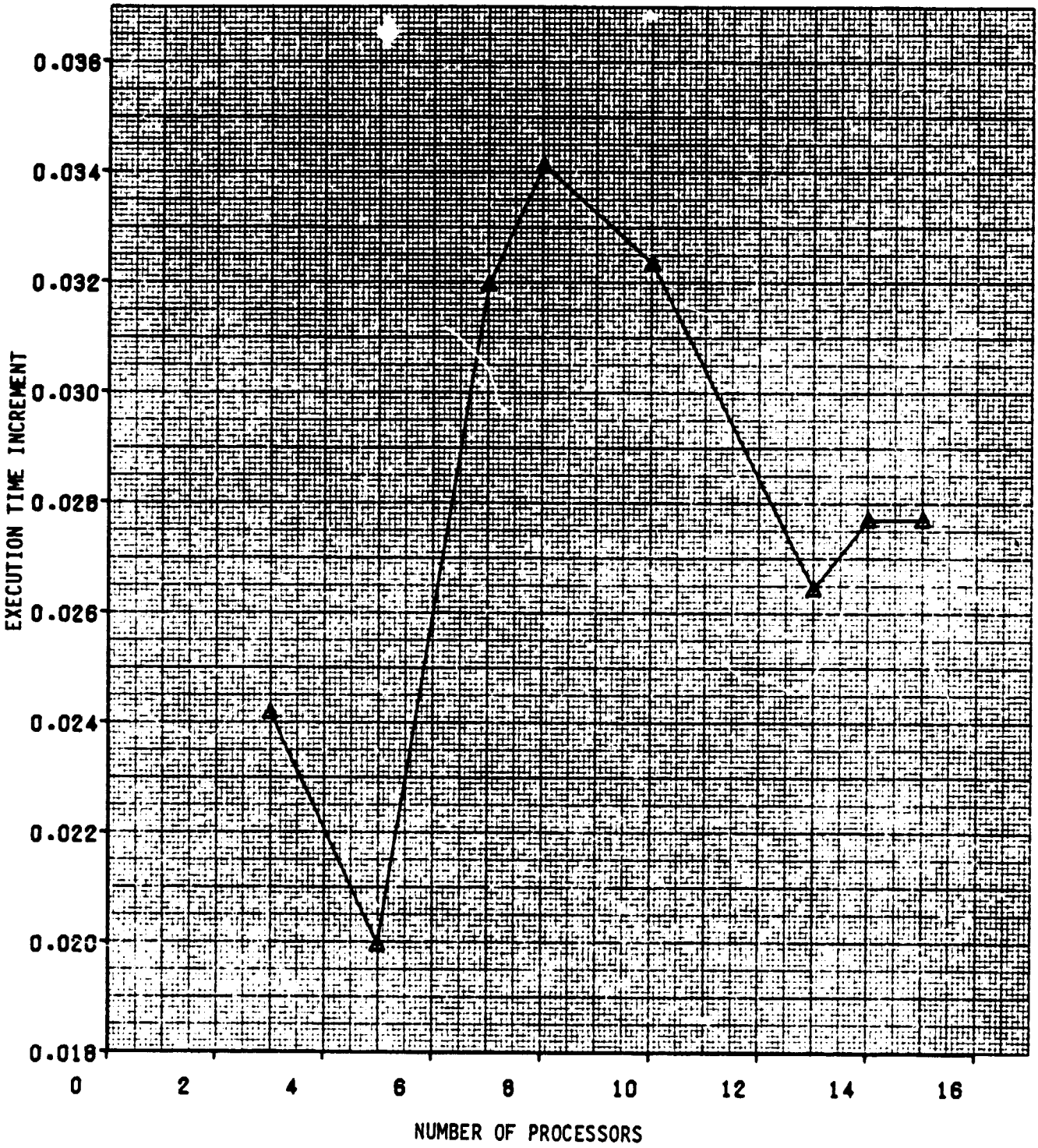


Figure 14(b) PPEM Configuration Summary

PPEM CONFIGURATION SURVEY
SYNCRONOUS DATA TRANSFER
SIMULTANEOUS BROYDEN ALGORITHM
DISTRIBUTED MATRIX OPTION

▲ ■■■ PARALLEL PROCESSOR ENGINE MODEL ■■■ S677729VU1 (ST 1

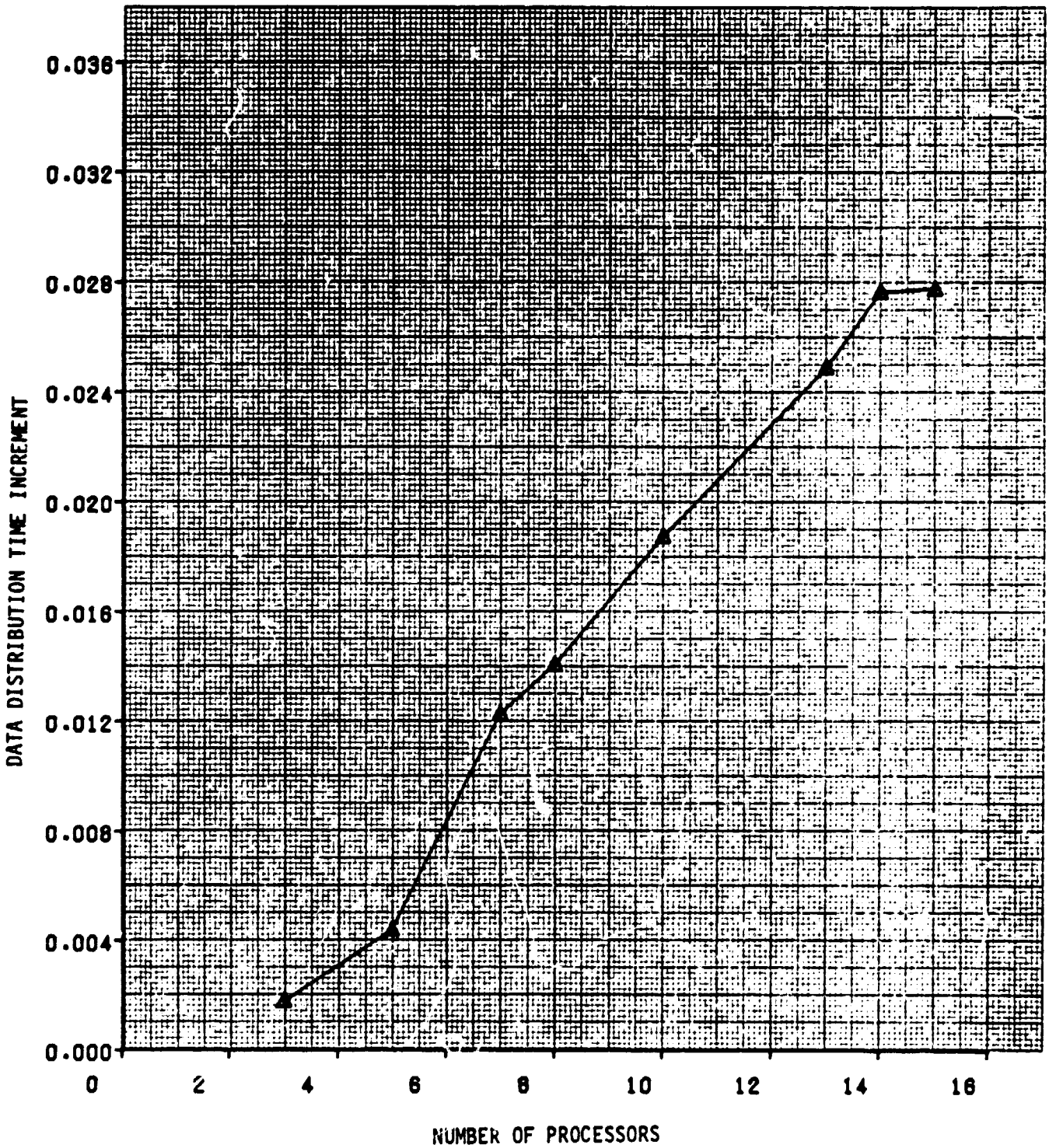


Figure 14(c) PPEM Configuration Summary

ORIGINAL PAGE IS
OF POOR QUALITY

PPEM CONFIGURATION SURVEY
ASYNCHRONOUS DATA TRANSFER
SIMULTANEOUS BROYDEN ALGORITHM
RESIDENT MATRIX OPTION

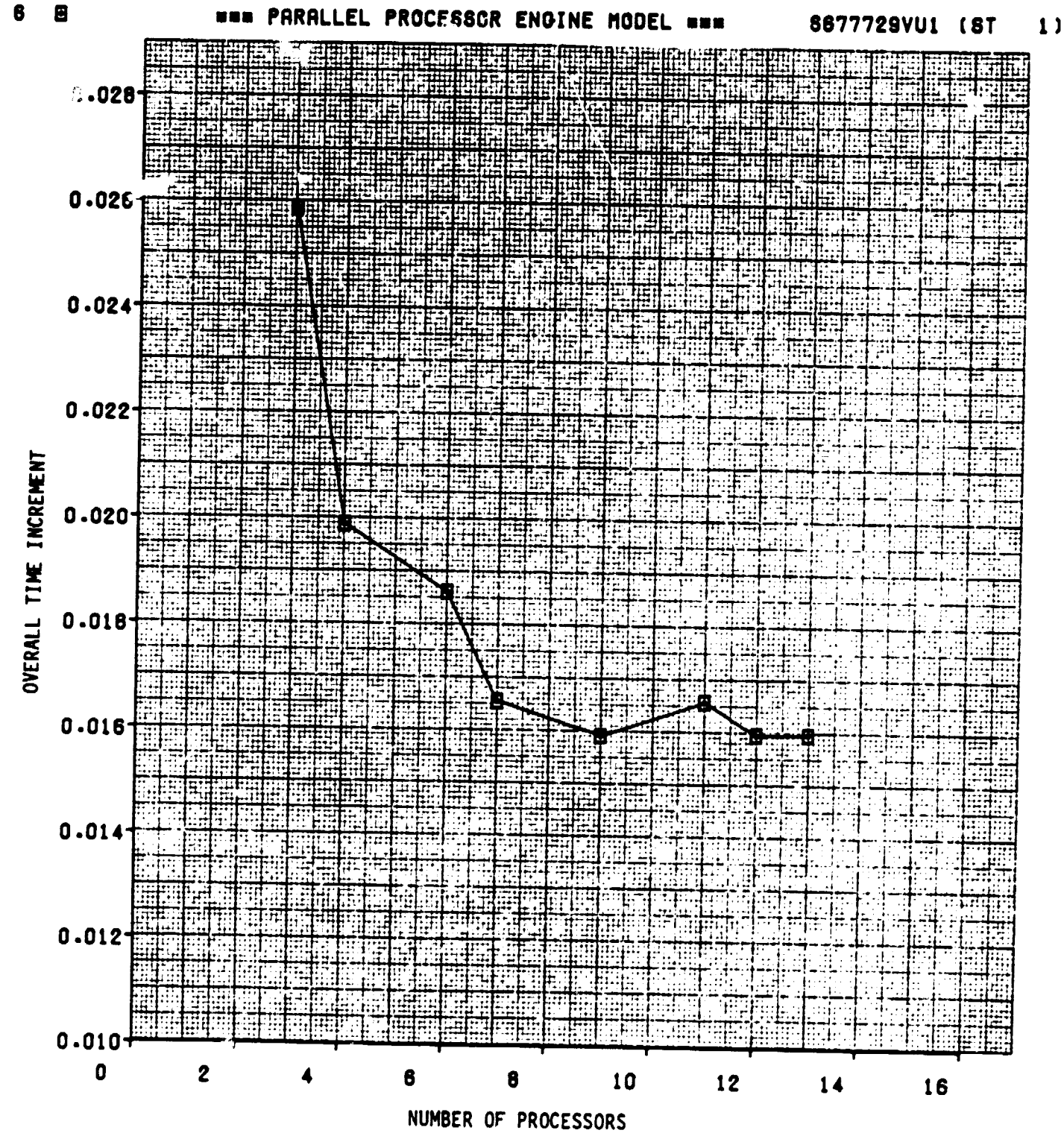


Figure 14(d) PPEM Configuration Summary

PPEM CONFIGURATION SURVEY
ASYNCRONOUS DATA TRANSFER
SIMULTANEOUS BROYDEN ALGORITHM
RESIDENT MATRIX OPTION

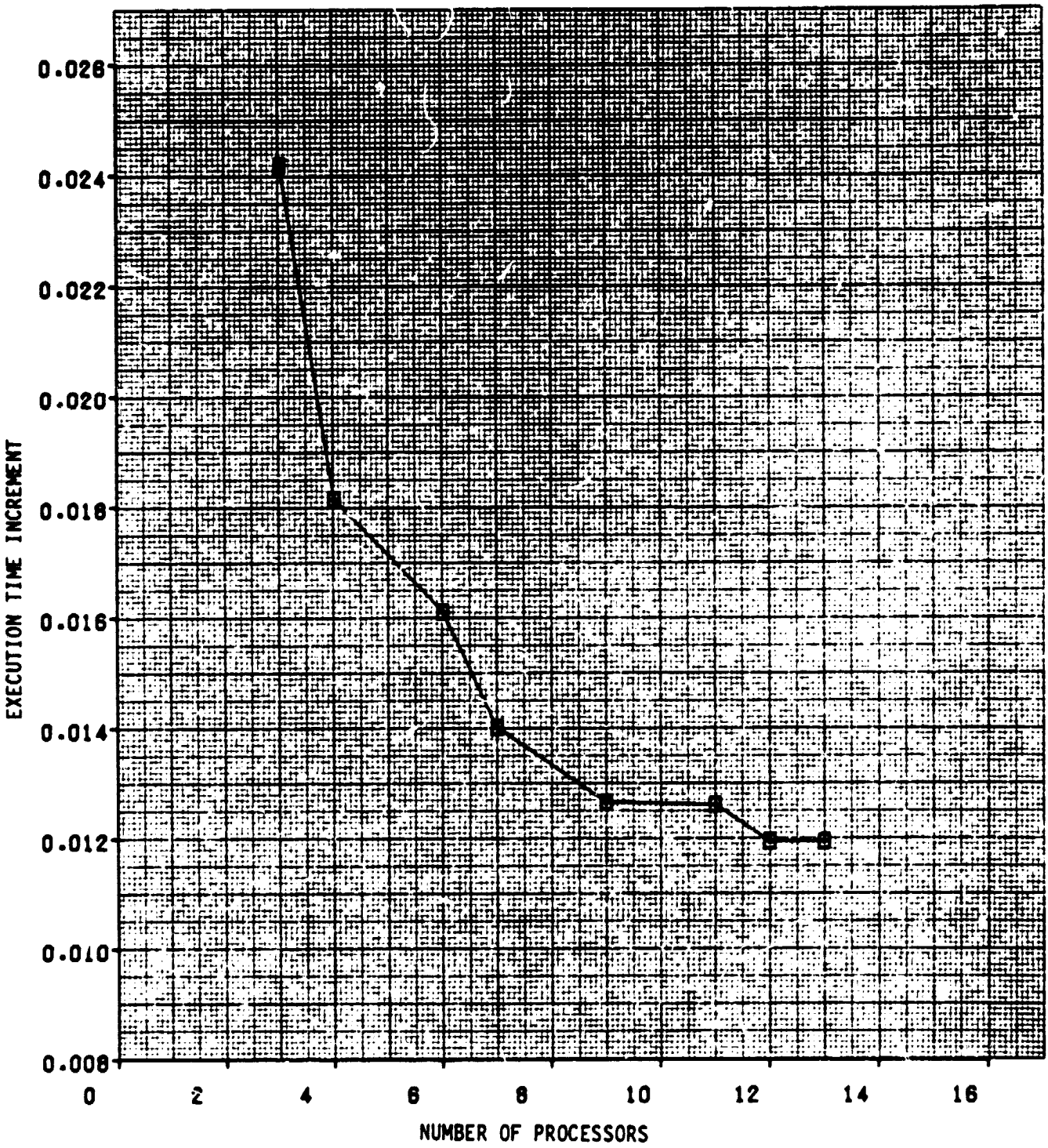


Figure 14(e) PPEM Configuration Summary

ORIGINAL PAGE IS
OF POOR QUALITY

PPEM CONFIGURATION SURVEY
ASYNCRONOUS DATA TRANSFER
SIMULTANEOUS BROYDEN ALGORITHM
RESIDENT MATRIX OPTION

6 B ■■■ PARALLEL PROCESSOR ENGINE MODEL ■■■ 8677729VU1 (ST 1)

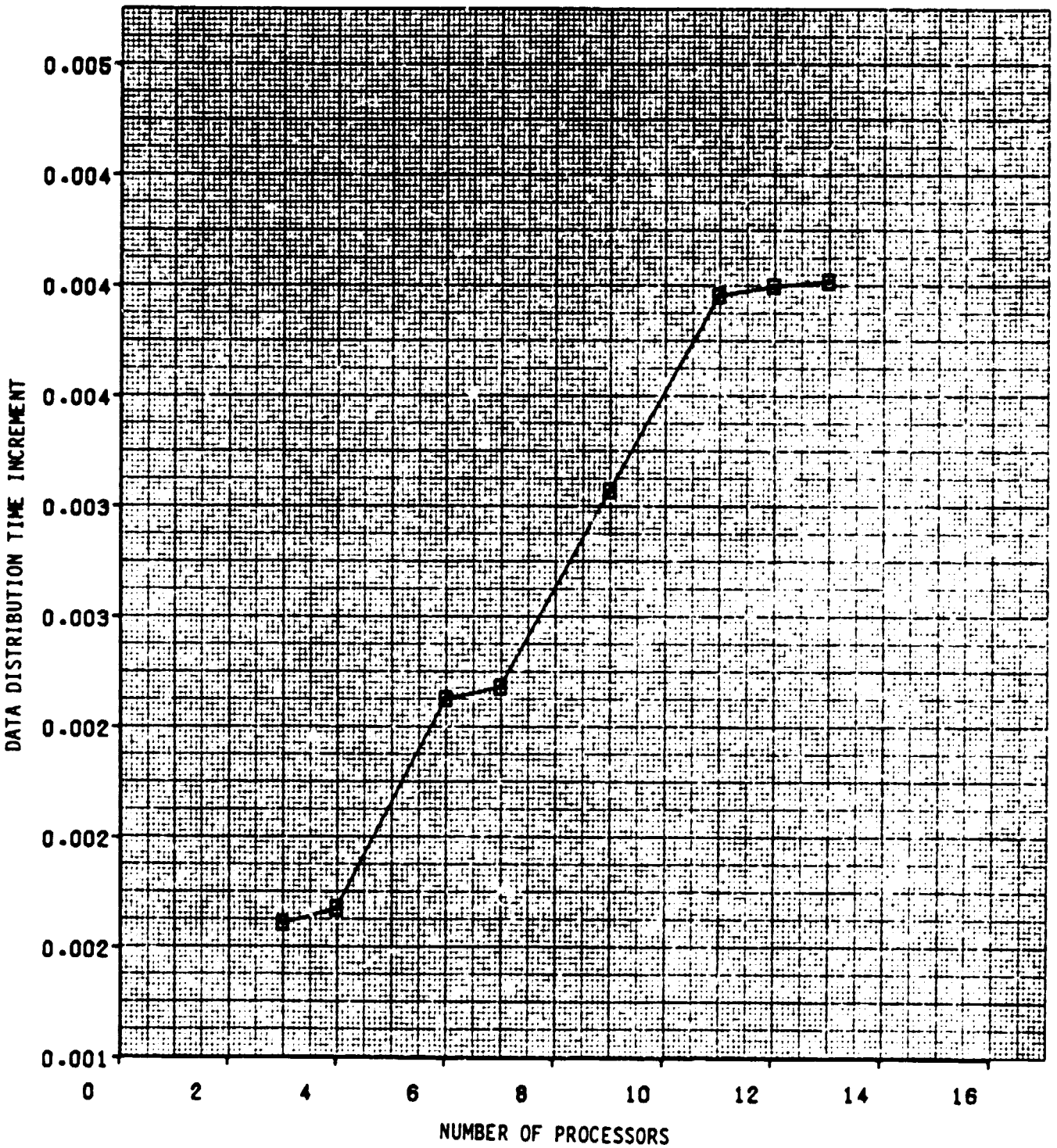


Figure 14(f) PPEM Configuration Summary

Accuracy

The effects of time increment and precision on the Figure-of-Merit (FOM) defined below were determined using the base configuration. The FOM is computed as an average root-mean-square error (RMSERR) made up of contributions from each of the individual error terms generated by the model.

$$RMSERR = \sqrt{\sum_{N} (ERROR)^2 / N}$$

$$FOM = \sum_{T} \frac{(RMSERR \times DT)}{\sum_{T} DT}$$

Where:

ERROR	-an individual error term
N	-number of iteration variables
DT	-time increment
T	-transient time

The effect of the size of time increment on the level of error associated with the real-time technique is documented in Figure 15. As can be seen, there is little change in the FOM, as the time increment is reduced below .025 seconds, however, the FOM does begin to show some increase at the largest stable time increment (.035 seconds).

The use of time increments greater than .035 seconds results in unstable operation and failures during transient, low power operation. At this point, the ratio of time increment to smallest time constant, a measure of the simulation's stiffness, is well over 100. This far exceeds the capability of any other explicit technique.

It is possible that the unstable responses found at larger time increments are more related to model quality than to limitations of the technique. In fact, this explicit technique may be A-stable, within the applicable range of the nonlinear model. Experiments utilizing small control input perturbations yield stable responses for a time increment of 1 second, which is the maximum value allowed by its scale factor. The results of this experiment are in line with operation in the fully converged steady-state mode. The process by which non-disturbed transients utilizing very large time increments converge is numerically identical to the steady-state mode, which is A-stable. Therefore, the real-time technique can be said to be similary stable, although model inconsistencies will be more likely to cause transient failures as the time increment is increased.

The randomness introduced by the quality of the model utilized is illustrated by the data point at a time increment of .025 in Figure 15. When runs are executed with slightly different increments near .025, an FOM of approximately .003 is obtained, more in line with the rest of the data. Examination of the run which produced the higher FOM reveals a much greater error during a particular portion of the transient than is evident in other similar runs.

Fifteen - bit arithmetic precision was assumed for the cases discussed above. Greater precision is more costly in terms of update rate, so that the capability of the technique to tolerate relatively low-precision operations is an important characteristic recommending its use. Figure 16 shows the effects of varying arithmetic precision. Twenty-four bit precision represents typical 32-bit floating point precision, while 30-bit precision is afforded by the use of scaled 32-bit integer operations. As can be seen, some improvement results in the FOM as precision is increased, but it is unlikely that the use of slower 32-bit operations can be justified on this basis as long as execution rate is the limiting factor.

The FOM described above is an appropriate method for comparing the relative accuracy of the real-time technique. In order to validate this technique, however, it is necessary to compare its results with those obtained from a source not affected by the residual errors associated with the real-time technique. This source is a version of the simulation in which the iteration errors are fully converged. This is the normal form of the general Gear-Broyden method, of which the real-time method is a special case.

By comparing the responses obtained from the two sources, a qualitative appreciation of the accuracy of the real-time technique is obtained. Since the FOM measured from the fully converged method is negligible in comparison to that obtained using the real-time technique, the effect on the real-time responses that produce a typical FOM can be visualized.

The data shown in Figures 17a through 17n was obtained from the same configuration as described earlier. The fully converged version employed the identical iteration variables as that required for the parallel, real-time emulation. Comparison of the two responses shows that they are nearly identical.

Over certain intervals of time, the real-time technique exhibits oscillatory instability which is not evident in the fully converged results. This phenomenon is virtually certain to be the result of local model discontinuities, probably in the turbine flow parameter map data. Similarly, the response following discontinuous changes in the control fuel flow exhibits higher-than-average errors. In both instances, the corrections provided by the Broyden update take place over several increments of time; and, while that is occurring, the residual iteration errors will be larger than normal.

The most important aspect of the results shown is that for responses in the frequency spectrum associated with gross power changes in the turbofan engine, the real-time technique provides adequate accuracy. Localized errors in the responses to very rapid or noisy control system inputs are expected in any real-time method . The key to the technique's capability is that these responses are bounded and, in fact, damp out very quickly. The self-correction provided by the Broyden algorithm provides much more reliable results than other explicit methods operating on stiff, nonlinear engine models.

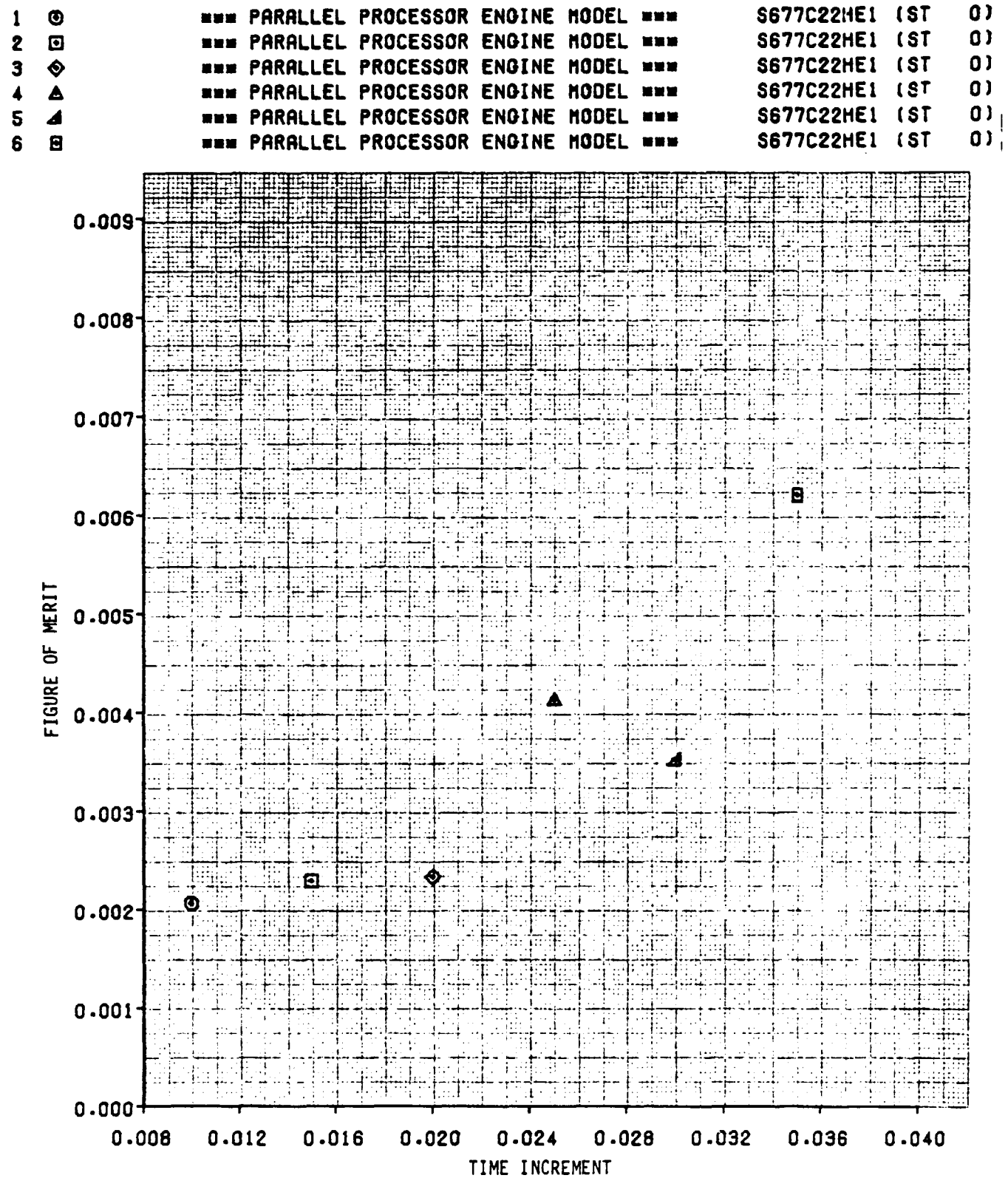


Figure 15 Error Measured by the Figure-of-Merit as a Function of the Time Increment

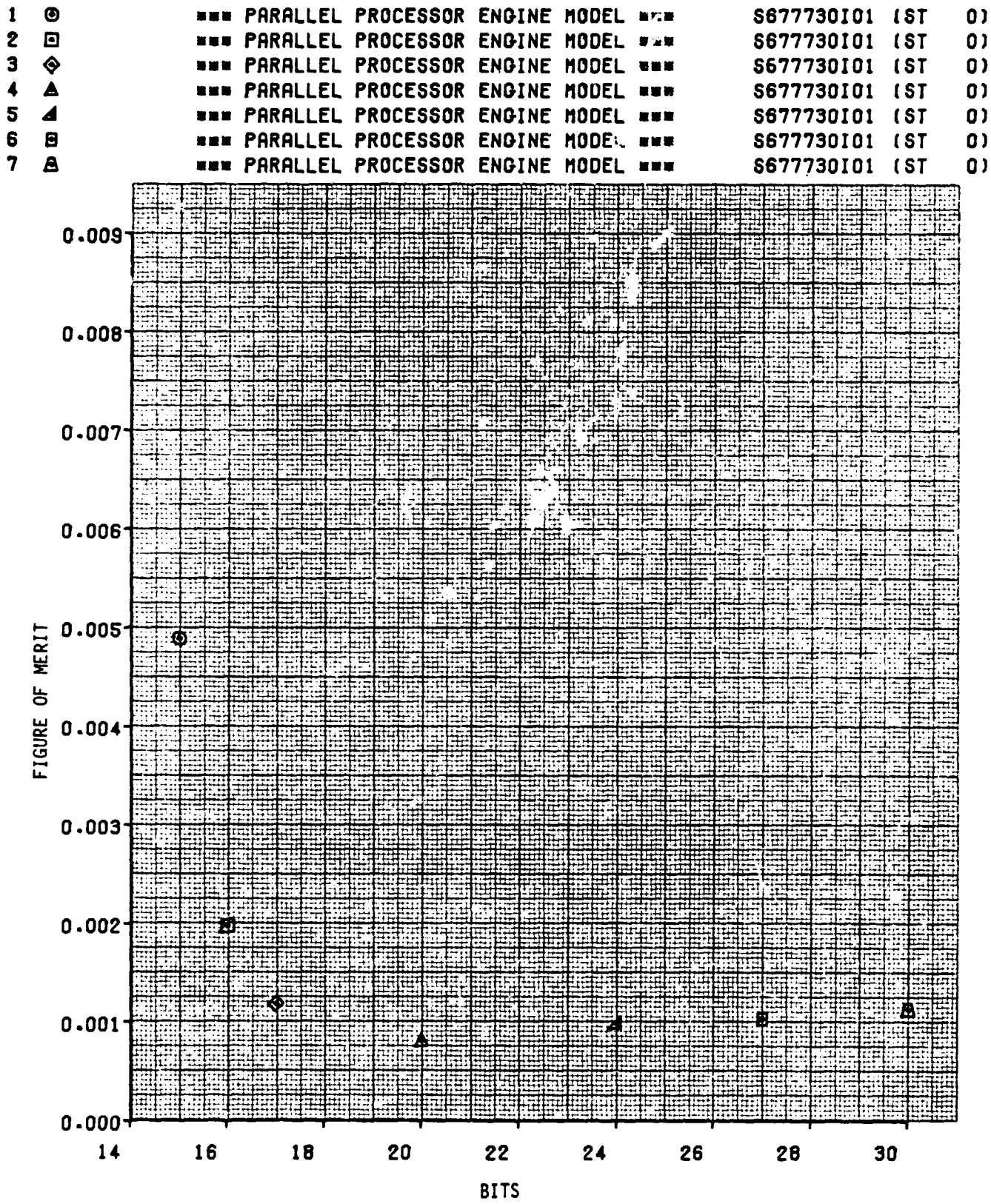


Figure 16 Figure-of-Merit as a Function of Precision

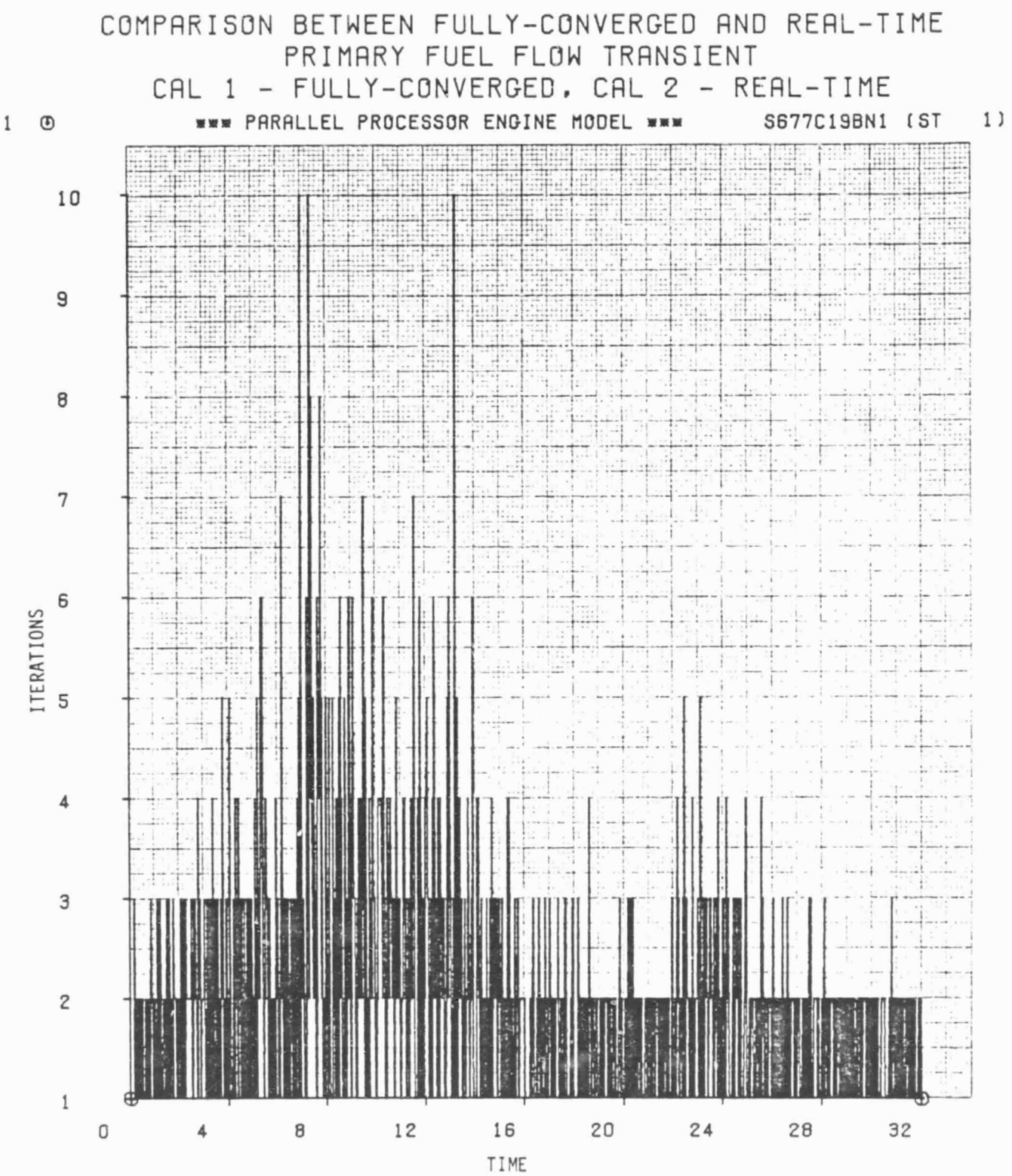


Figure 17(a) Comparison Between Fully Converged and Real-Time Results

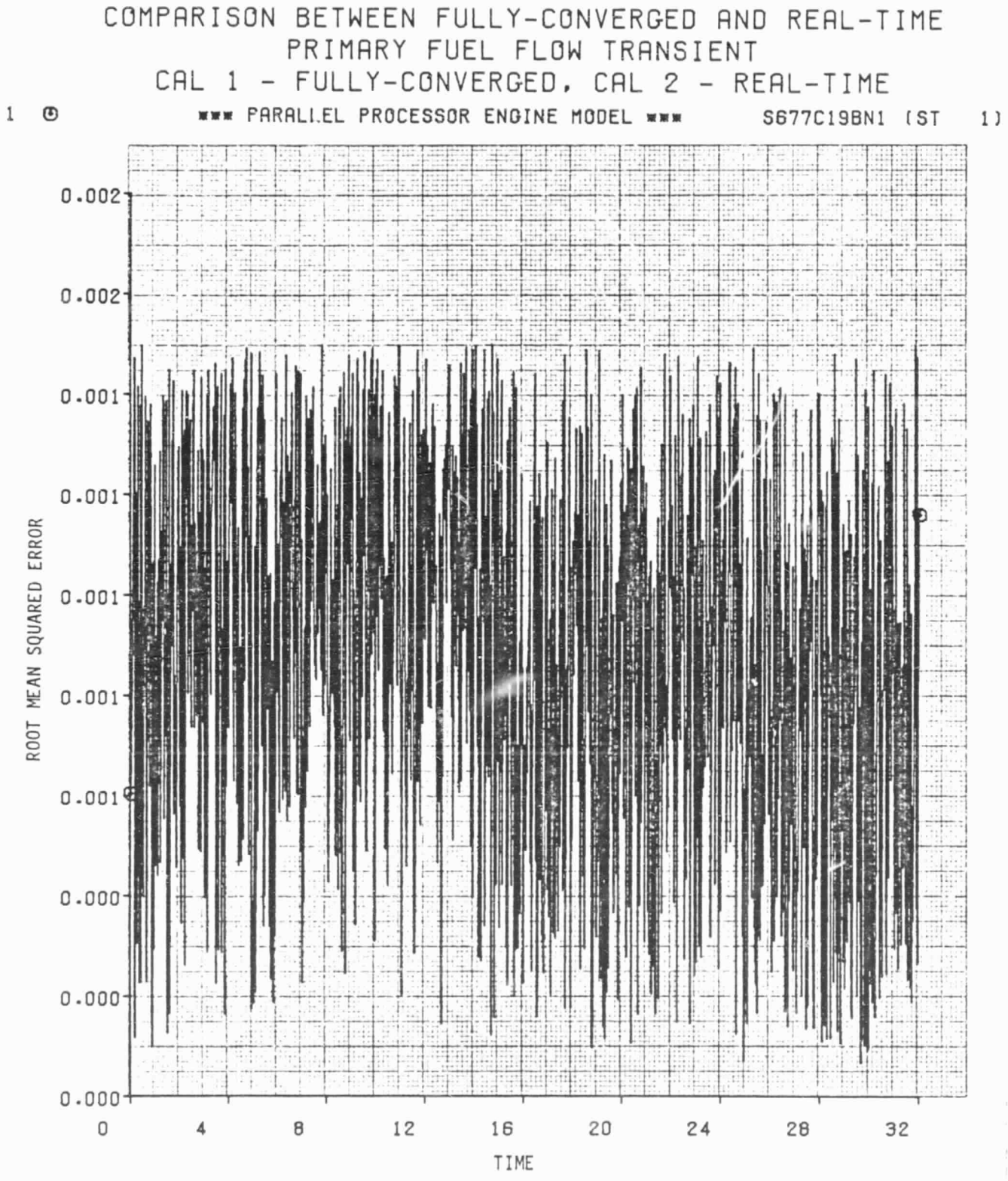


Figure 17(b) Comparison Between Fully Converged and Real-Time Results

COMPARISON BETWEEN FULLY-CONVERGED AND REAL-TIME
PRIMARY FUEL FLOW TRANSIENT
CAL 1 - FULLY-CONVERGED, CAL 2 - REAL-TIME

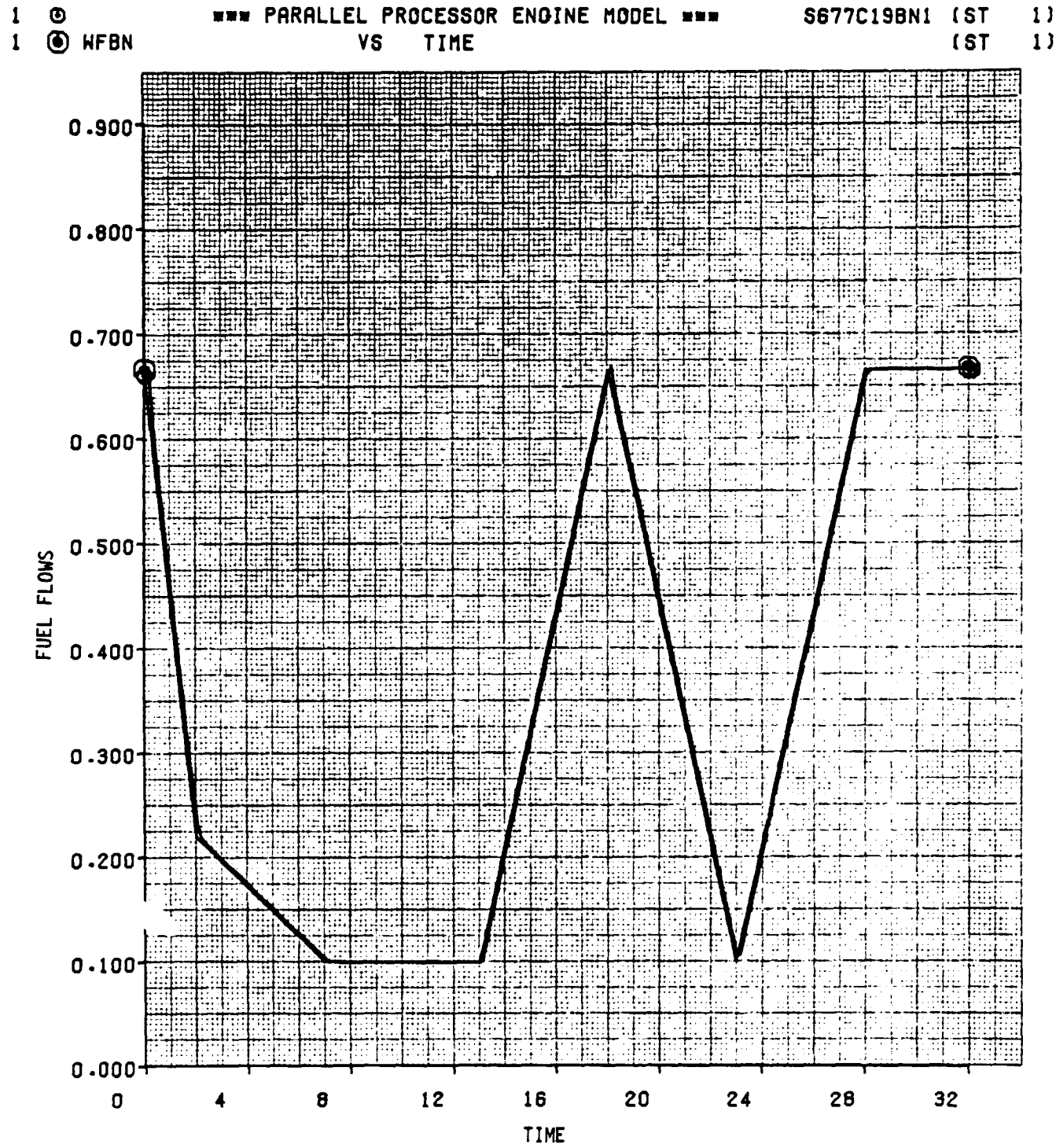


Figure 17(c) Comparison Between Fully Converged and Real-Time Results

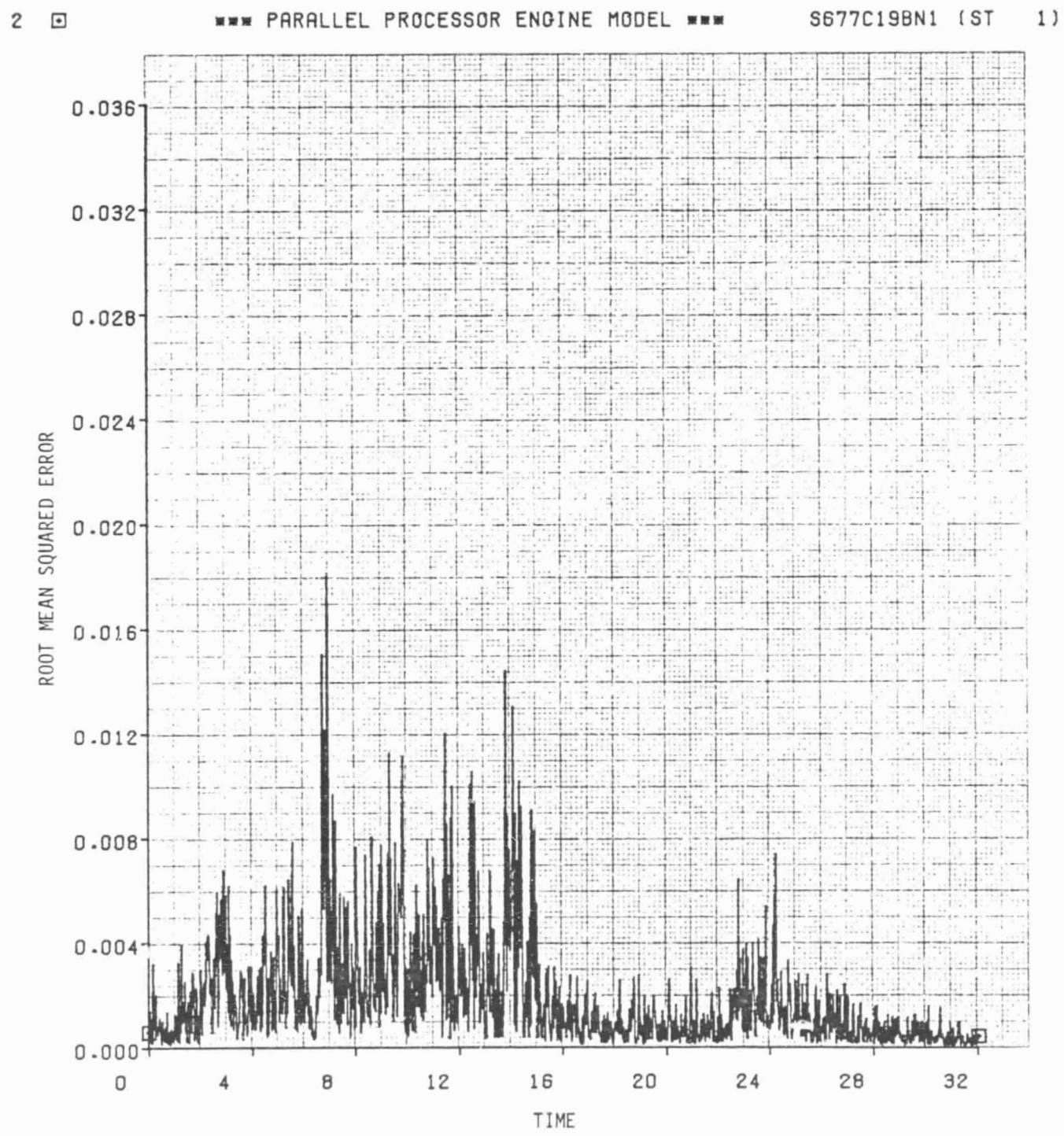


Figure 17(d) Comparison Between Fully Converged and Real-Time Results

COMPARISON BETWEEN FULLY-CONVERGED AND REAL-TIME PRIMARY FUEL FLOW TRANSIENT

CAL 1 - FULLY-CONVERGED, CAL 2 - REAL-TIME

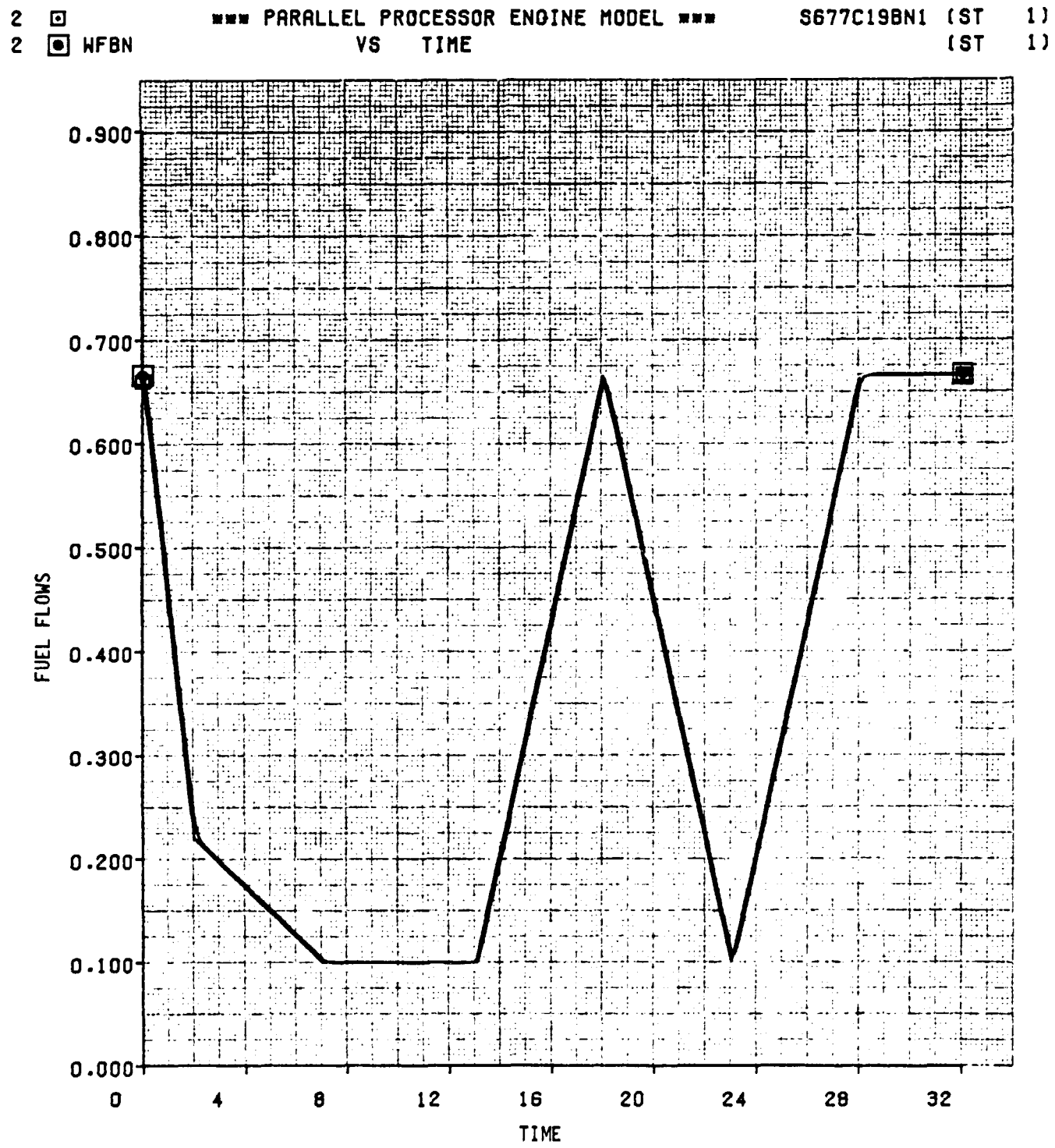


Figure 17(e) Comparison Between Fully Converged and Real-Time Results

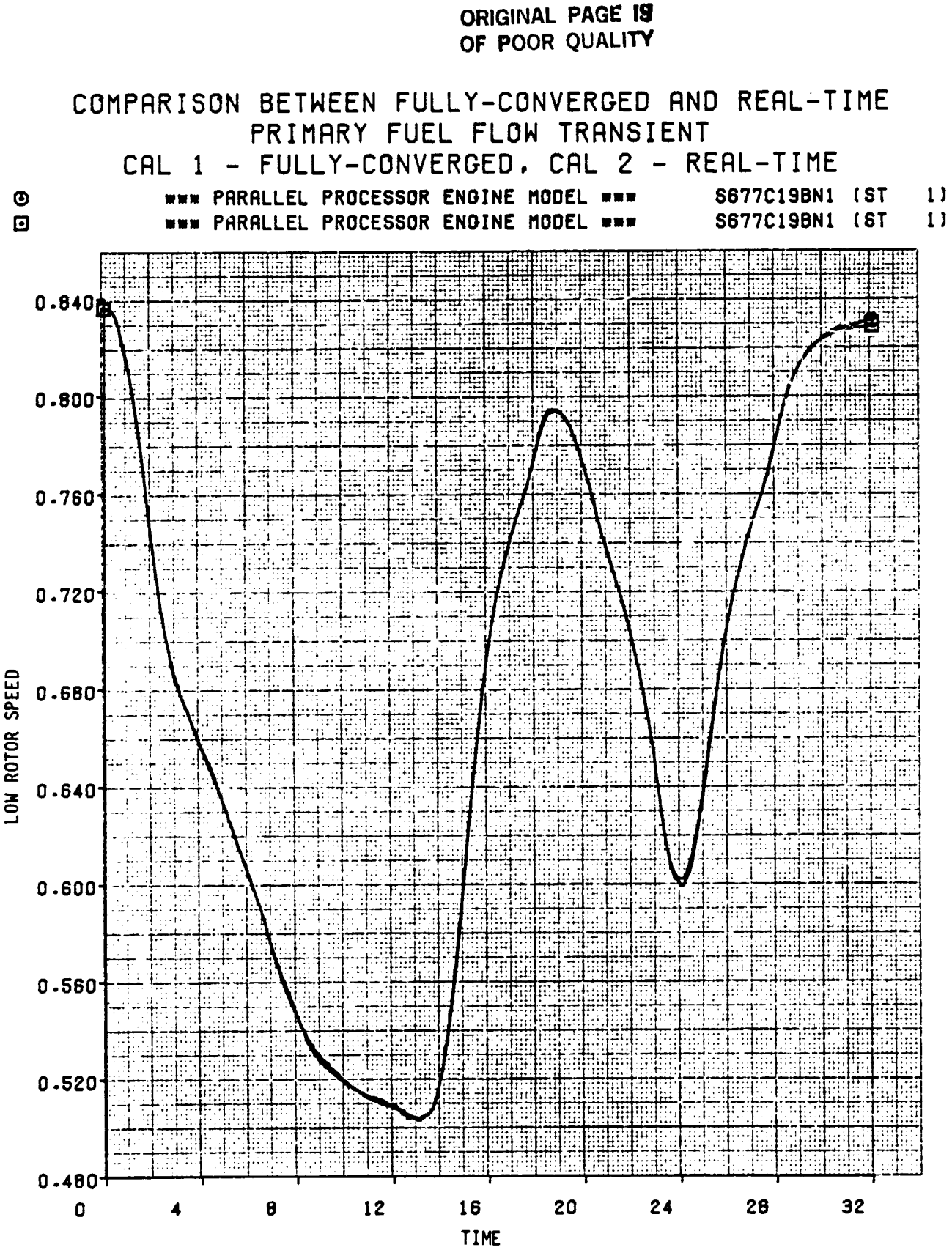


Figure 17(f) Comparison Between Fully Converged and Real-Time Results

COMPARISON BETWEEN FULLY-CONVERGED AND REAL-TIME PRIMARY FUEL FLOW TRANSIENT

CAL 1 - FULLY-CONVERGED, CAL 2 - REAL-TIME

1 ⊕ *** PARALLEL PROCESSOR ENGINE MODEL *** S677C19BN1 (ST 1)
2 ⊡ *** PARALLEL PROCESSOR ENGINE MODEL *** S677C19BN1 (ST 1)

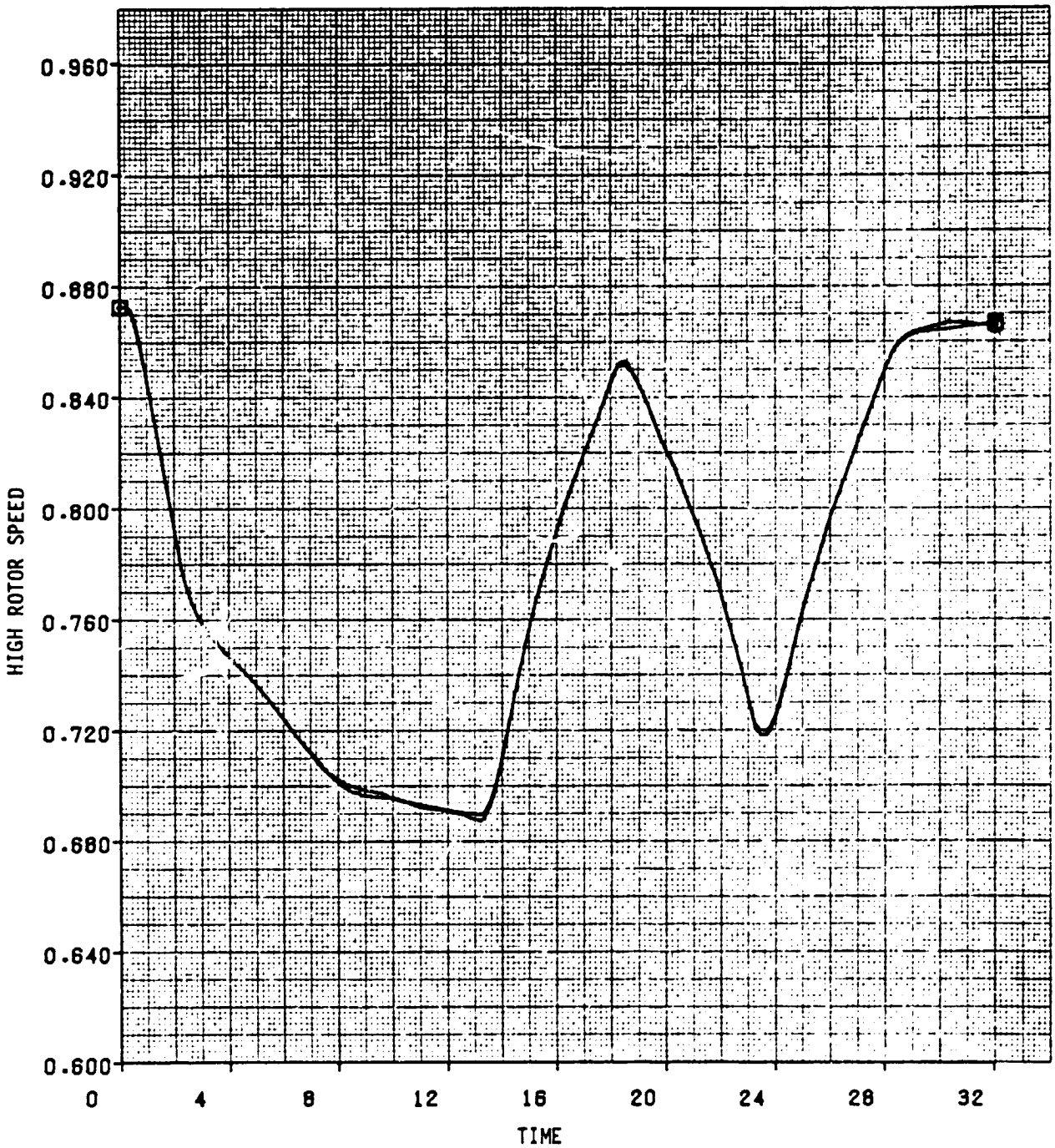


Figure 17(g) Comparison Between Fully Converged and Real-Time Results

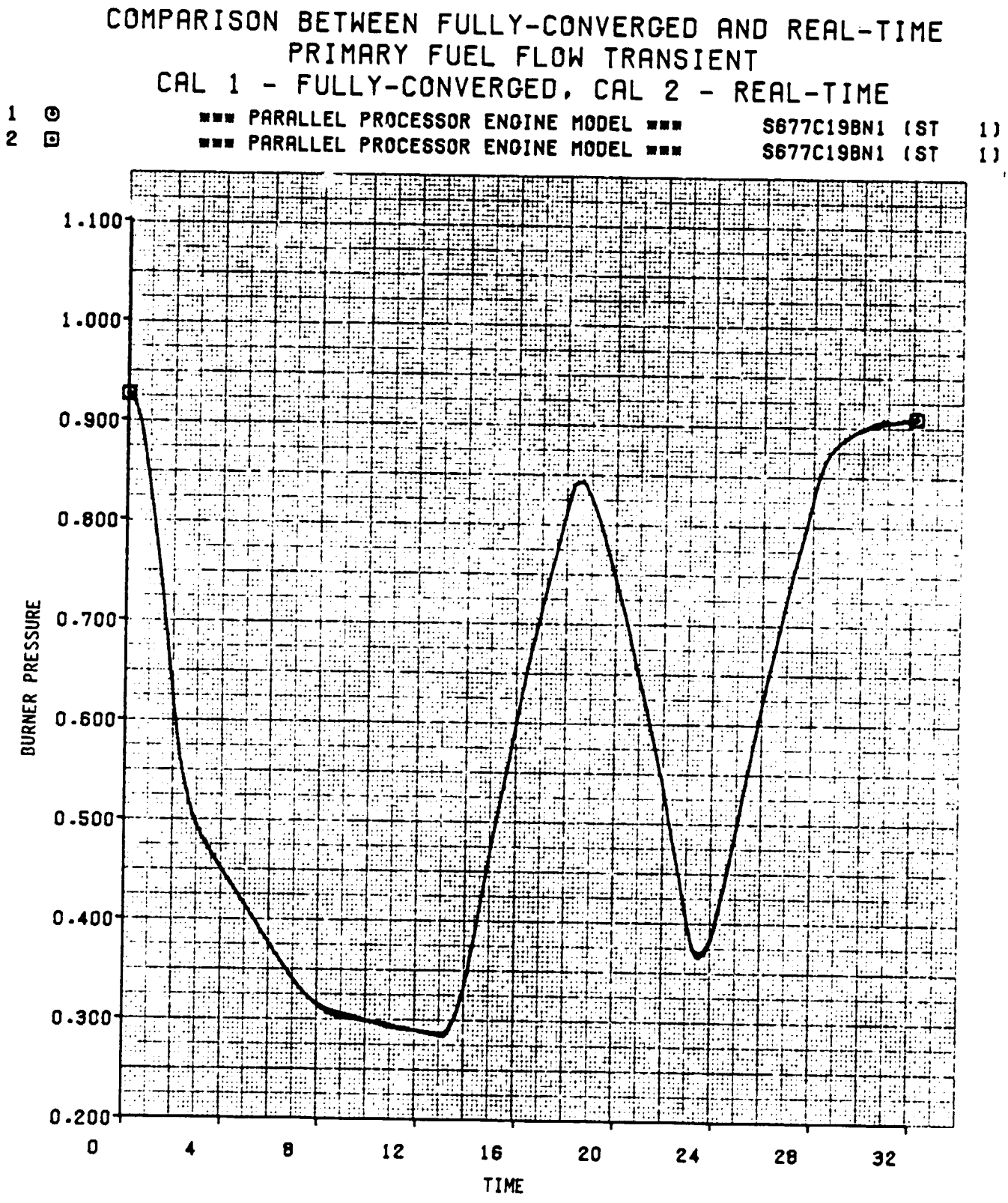


Figure 17(h) Comparison Between Fully Converged and Real-Time Results

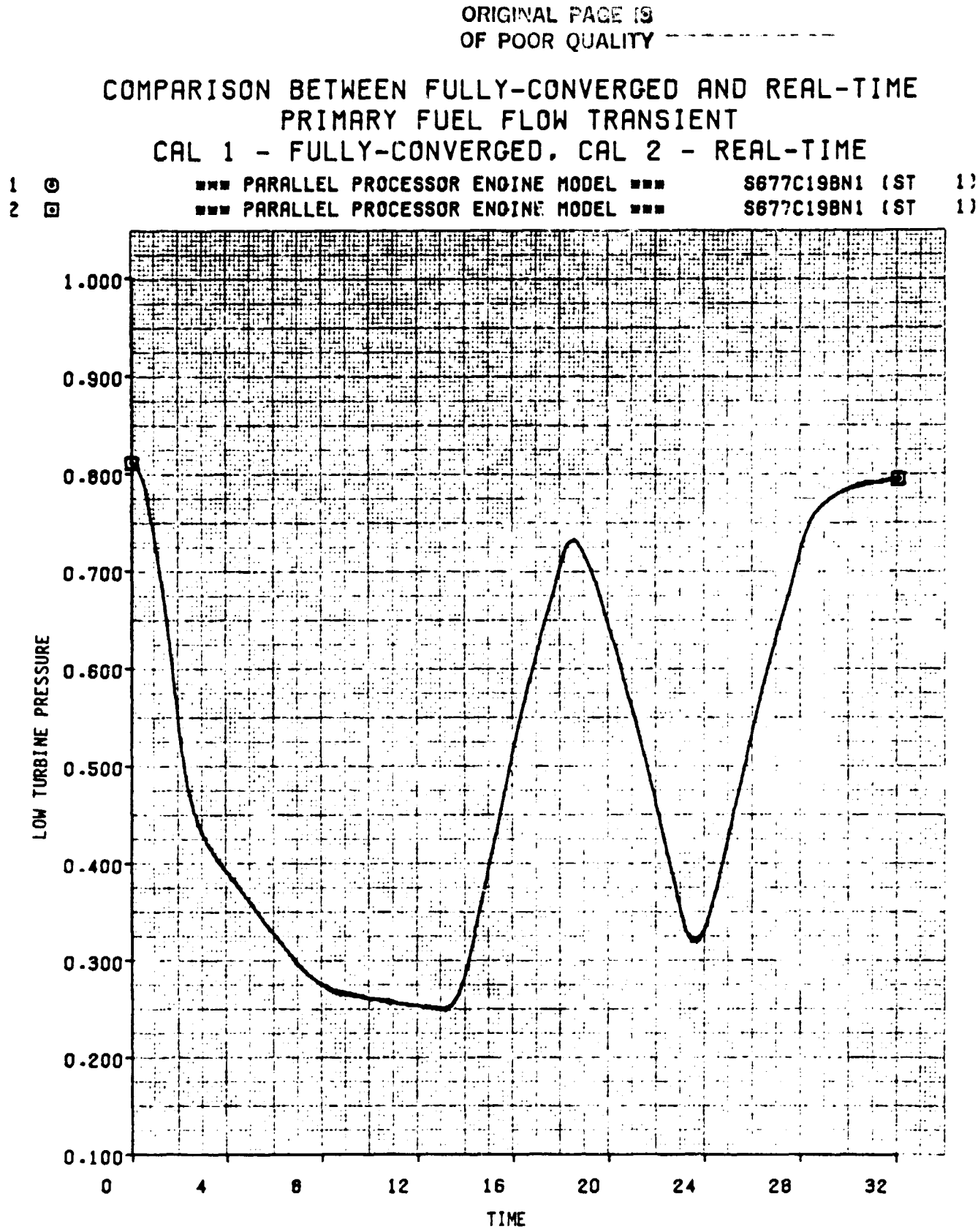


Figure 17(i) Comparison Between Fully Converged and Real-Time Results

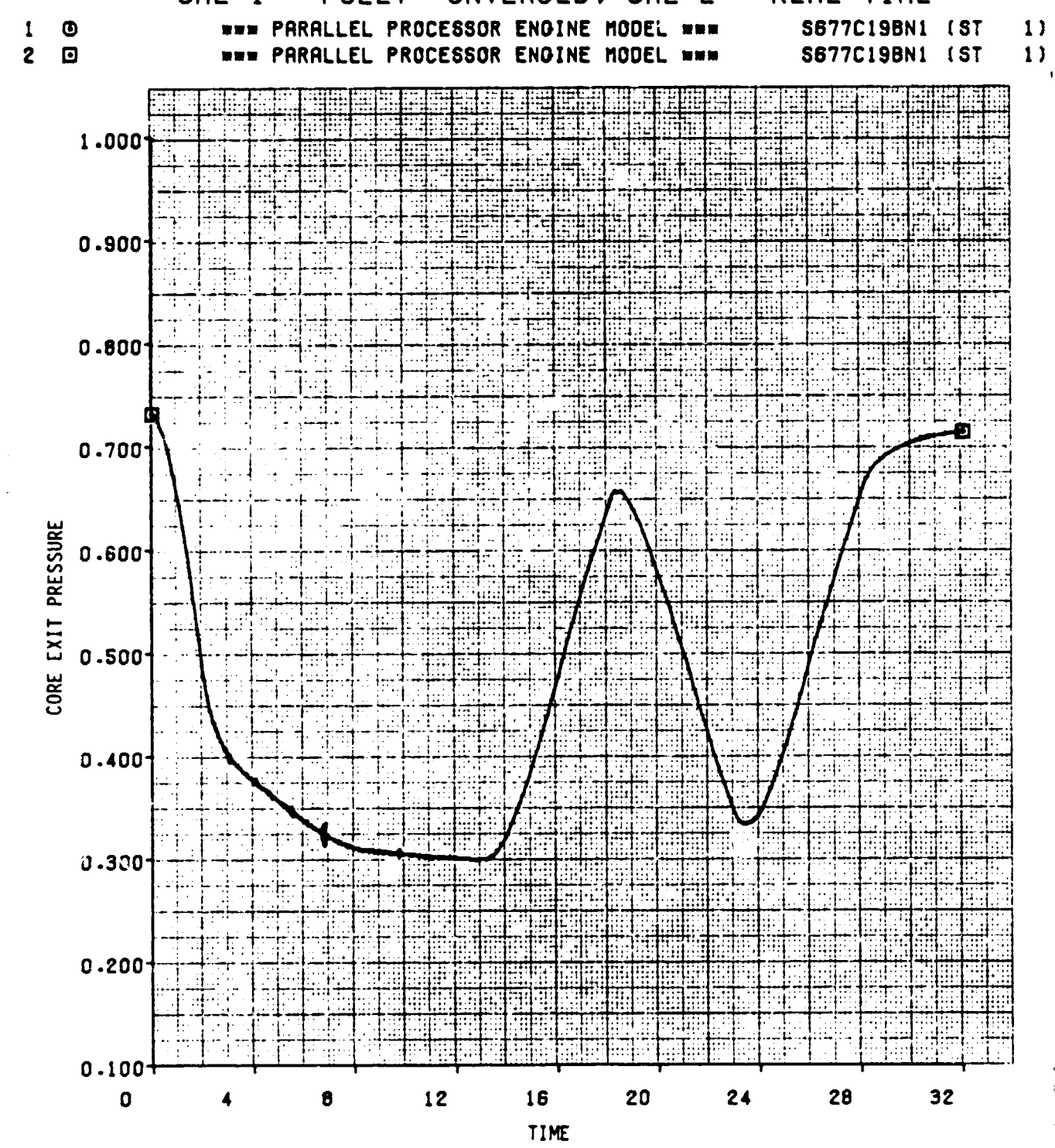


Figure 17(j) Comparison Between Fully Converged and Real-Time Results

COMPARISON BETWEEN FULLY-CONVERGED AND REAL-TIME PRIMARY FUEL FLOW TRANSIENT

CAL 1 - FULLY-CONVERGED, CAL 2 - REAL-TIME

1 ⊙ *** PARALLEL PROCESSOR ENGINE MODEL *** S677C19BN1 (ST 1)
2 ⊡ *** PARALLEL PROCESSOR ENGINE MODEL *** S677C19BN1 (ST 1)

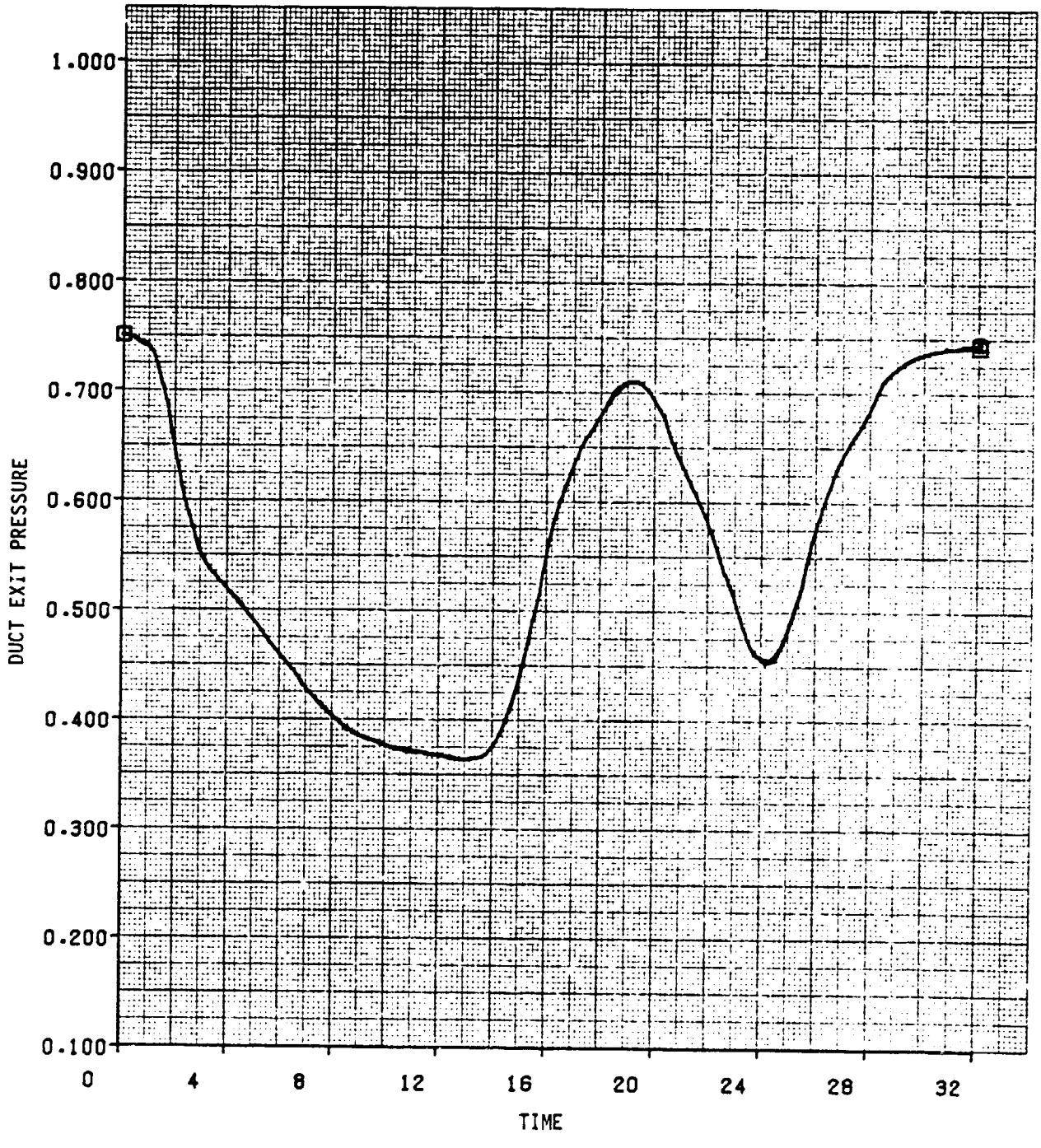


Figure 17(k) Comparison Between Fully Converged and Real-Time Results

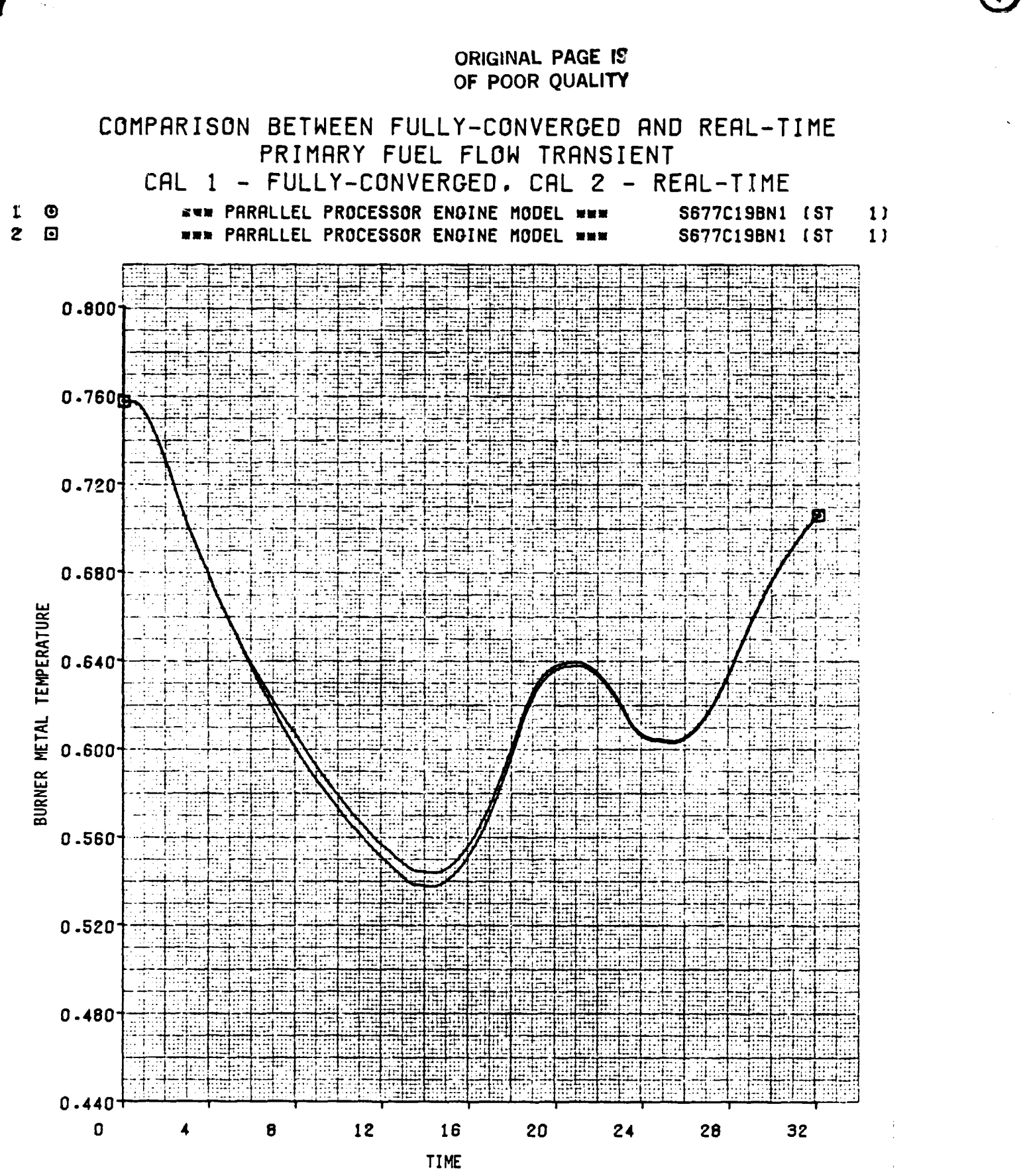


Figure 17(1) Comparison Between Fully Converged and Real-Time Results

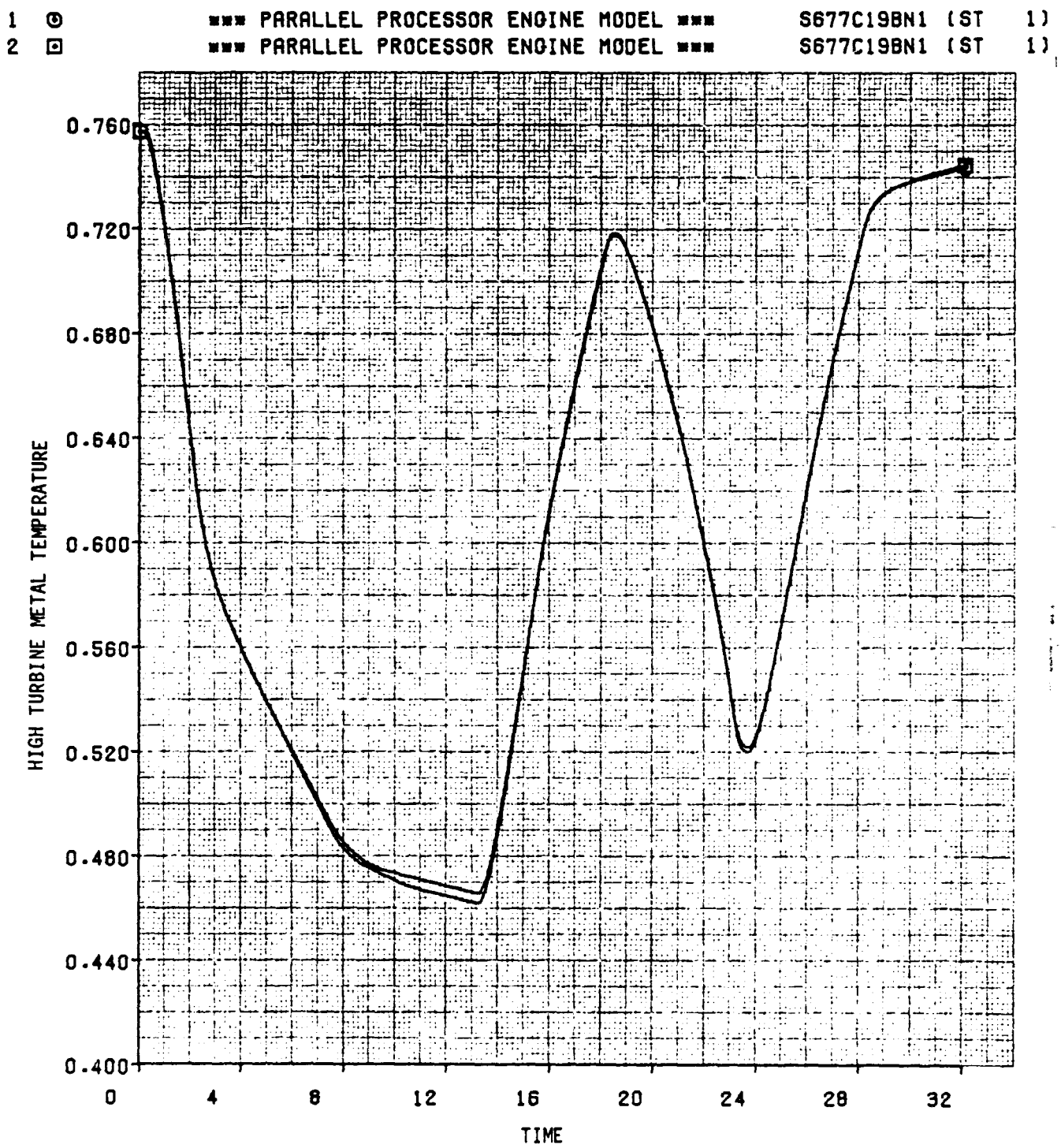


Figure 17(m) Comparison Between Fully Converged and Real-Time Results

COMPARISON BETWEEN FULLY-CONVERGED AND REAL-TIME
PRIMARY FUEL FLOW TRANSIENT
CAL 1 - FULLY-CONVERGED, CAL 2 - REAL-TIME

1 ⊙ *** PARALLEL PROCESSOR ENGINE MODEL *** S677C19BN1 (ST 1)
2 ⊡ *** PARALLEL PROCESSOR ENGINE MODEL *** S677C19BN1 (ST 1)

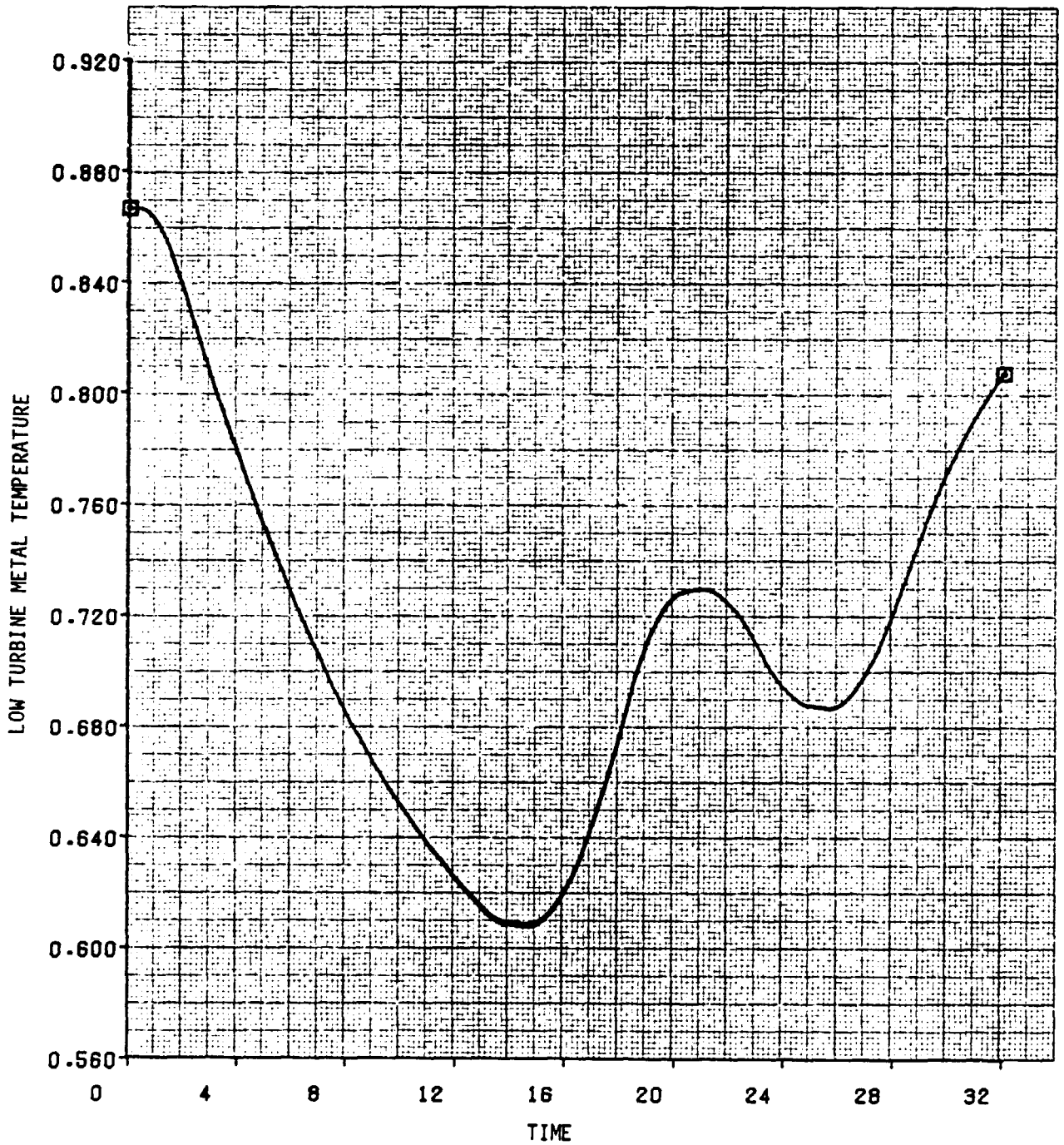


Figure 17(n) Comparison Between Fully Converged and Real-Time Results

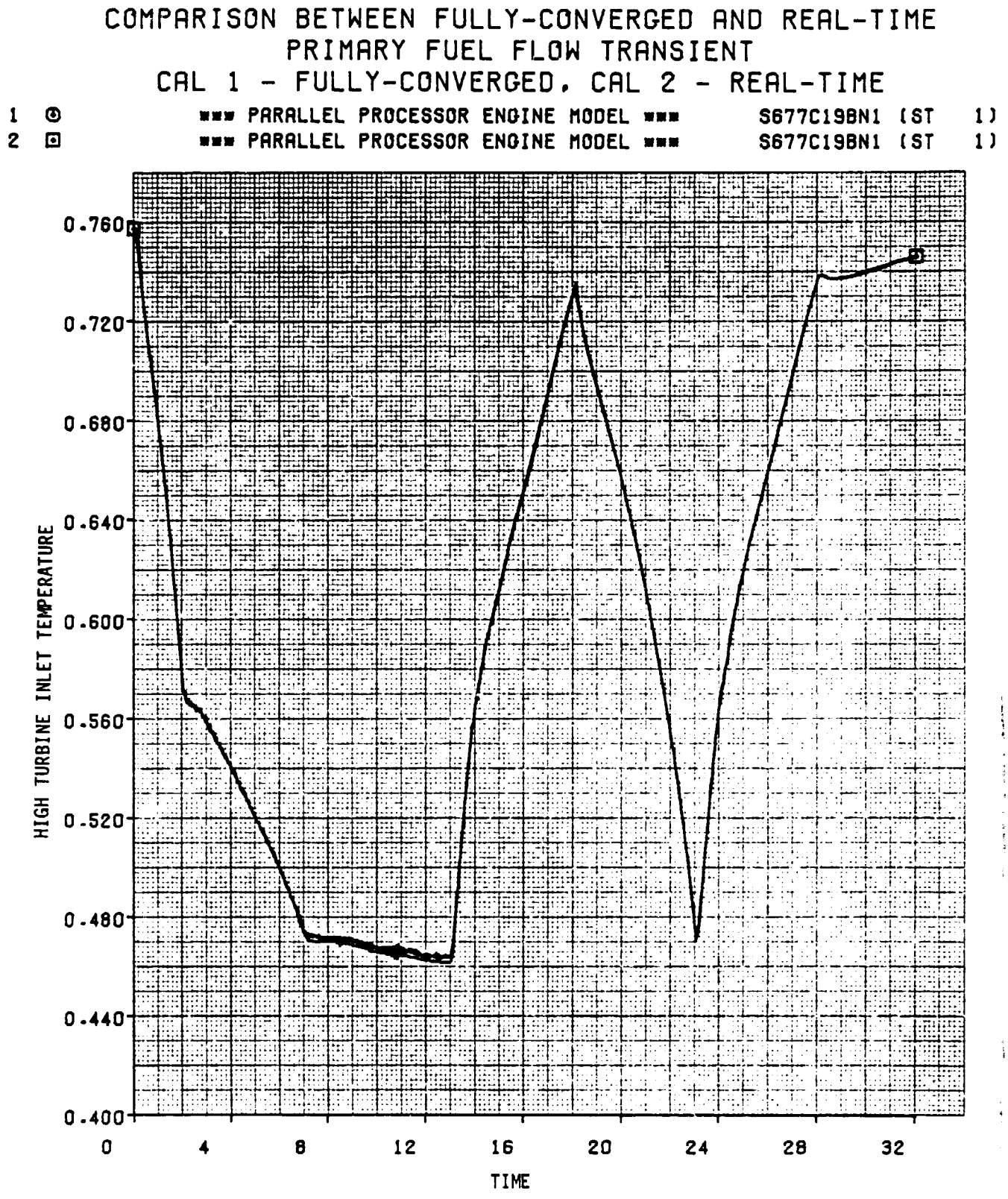


Figure 17(o) Comparison Between Fully Converged and Real-Time Results

Use of the Emulation Program to Estimate Cost Comparisons Between Parallel and Serial Processing

The reduction in required time increment demonstrated with the parallel real-time technique is offset by the need for several, rather than one, processors. In addition, the increased complexity of parallel software will result in greater cost. Eliminating this nonrecurring expense from consideration, a comparison can be made based on an estimate of the cost of greater execution rate capability if serial processors are utilized. If the cost of parallel processing is taken to be proportional to the number of processors required to achieve a given speed-up factor, then the ratio of serial to parallel cost can be obtained and plotted versus the speed-up factor. Figure 18 shows a typical representation of this cost comparison based on results obtained from the program. It has been assumed that the cost of faster serial processors is proportional to the ratio of the execution rates squared. From this data, it can be seen that the use of asynchronous data transfer is considerably more cost-effective than synchronous.

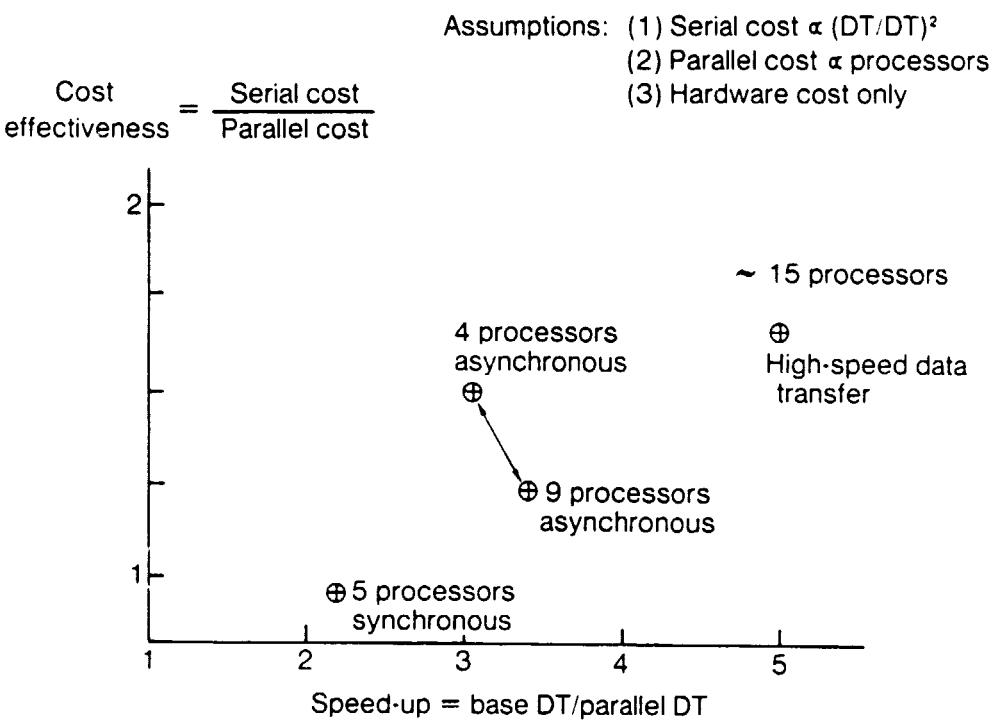


Figure 18 Cost Effectiveness Presentation

It appears that the limiting speed-up factor achievable with the parallel technique occurs at around five, utilizing fifteen or so processors. The cost effectiveness of this approach might be in the range of 1-1/2 to perhaps 2 in an advanced application, where a serial approach would be even more costly than the square law used to produce the figure.

Based on these considerations, it appears that the most cost-effective application of the parallel technique would be in systems requiring relatively high update rates or the ability to perform other computational tasks on the same processors used by the model. The former might include real-time simulations supporting control system bench testing. The latter implementation might utilize part of the overall cycle time to execute the parallelized model and evaluation routines. The remainder of the time would be used for fault detection and control law execution. An array of 15 MC68000-type processors, operating at an overall update time of 20 milliseconds, would leave about 10 milliseconds on the 15 microprocessors for control tasks. This would be sufficient to support algorithms consistent with the level of accuracy of the engine model.

VIII. CONCLUSIONS

Summary

1) Parallel microprocessors employing synchronous data transfer are capable of providing real-time simulation capability for turbofan engine applications. A speed-up factor of over 100 percent can be achieved using five processors.

2) The real-time simulation technique utilized is capable of stable operation when implemented on NASA's RTMPS.

3) Asynchronous data transfer greatly enhances the real-time performance of the parallel technique. A speed-up factor of over 200 percent can be achieved using nine processors.

4) Sixteen-bit scaled-integer arithmetic provides adequate precision for the technique if the nonlinear model is consistently well defined.

5) The simulation technique utilized here in either its real-time or fully converged modes can be sucessfully parallelized to any degree desired if high-speed asynchronous data transfer between the processors is employed.

Sixteen-Bit Precision

Practical applications of the technology discussed in this report will require the use of 16-bit microprocessors, employing 32-bit internal architecture. Therefore, the question of the technique's ability to function stably when utilizing this kind of processor is crucial.

A few comments concerning the quality on the nonlinear model utilized in this program are appropriate. First, the scaling convention used does not fully exploit the 16-bit word. In fact, the values resulting could be termed signed 15-bit integers. The use of biases on all the variables would enable the recovery of that extra bit's worth of precision. As can be seen in Figure 16, this would be a significant improvement in the performance of the technique.

A second observation that can be made is that the inception of instability, at time increments larger than .035 seconds, does not occur until relatively low power operation is attained. The quality of the turbine and compressor flow maps in this regime is questionable, as they are based on practice at least ten years old. Using models based on more modern map-fitting and validation tools would have to result in improved stability characteristics.

Further enhancements to the technique are possible which would tend to improve the precision of the Broyden update algorithm, a key factor in the technique's numerical stability. While these would incur some penalty in execution time, the need for extreme reliability would dictate that all possible steps be taken to ensure it.

On the basis of the results obtained in this program, there is every reason to believe that 16-bit architectures can reliably support the parallel implementatin of the real-time technique, providing that a modelling practice suggested by the foregoing discussion is followed and adequate processor execution rates are employed to ensure small enough time increments.

In order to support this thesis, a comprehensive set of numerical experiments, employing simulations of real external disturbances, would have to be performed as an important part of the system certification process. Stability would have to be demonstrated under all possible sets of external effects.

Stability

The use of 15-bit precision tends to reduce the stability margin afforded by the use of 32-bit data representations. The characteristic failure mode associated with the technique is one in which nominally stable operation becomes unbounded in the vicinity of certain model inconsistencies. This property of the technique is related primarily to the Broyden algorithm. Under the influence of models that may be locally ill-conditioned and with limited precision available in real-time, the ratio of increments used to update the matrix may become indeterminate to a degree that causes a breakdown in the process. This can be prevented establishing a lower limit on the size of the denominator factor used in the matrix update. However, if this tolerance is set too large, the algorithm will fail to provide updated information to the matrix. In this event, model nonlinearities will eventually cause failures similar to those discussed above.

Dynamic Characteristics of the Technique

The accuracy and stability considerations discussed in the preceding sections certainly affect the operation of the real-time simulation in a closed-loop environment where it interfaces with hardware systems. However, the dynamic characteristics of the real-time simulation itself, viewed as an element of a closed-loop test bench, are of equal importance. The most significant characteristic of digital real-time simulation is the time delay associated with the discrete process involved. While relatively low-frequency model inputs experience little distortion due to the delay, frequencies approaching the simulation update rate generate considerable spurious responses. If the response of the system is substantially affected, erroneous tests can be executed--and, at worst, unstable system response results.

Generally, the delayed signals produced by the simulation are compensated by extrapolating their values ahead in time somewhat before being transmitted to the system. Predictive compensation does not eliminate the problem but does improve the dynamic accuracy of the model insofar as the real system is concerned.

In the synchronous parallel version of the real-time technique reported here, the delay imposed is increased to two full cycles due to the constraint that data is only transferred once per cycle. While approprite extra compensation will alleviate this problem at lower frequencies, it is probable that the stability characteristics of closed-loop systems employing this technique in its parallel implementation will not be improved, even though higher update rates are achieved. In any event, stability should not be a constraining factor as long as maximum update rates are correctly specified in the design of the closed-loop facility.

The use of asynchronous data transfer allows a return to the original, serial delay effect. This property, coupled with the much higher update rates resulting from the use of more processors, promotes this approach as a means of achieving extremely high update rates.

Limitations

The application of asynchronous data transfer to a large array of processors can yield, at best, the time increment required for the model plus the data transfer time. This represents the limiting case for the technique. For this model and a one-microsecond data transfer time, this value is not much less than that acheived with nine processors--around .015 seconds. The use of very high-speed data transfer devices coupled with an array of around 15 processors might yield a configuration that achieved a .01-second update increment in a cost-effective manner. Further reductions would be anticipated as faster hardware is available at costs comparable with the hardware used as the basis for this program.

The next generation of turbofan engine control systems will probably double the current response requirements. In general, update increments of around .005 seconds would be required for the fastest elements of models that might be utilized in advanced control modes. This translates into a requirement for an array of microprocessors numbering around 15 to 16, operating at processor clock rates of around 20 to 30 megahertz. This system would be capable of performing the full range of fault detection, scheduling, and compensation functions.

Future Applications of Parallel Processing in Simulation

The promise demonstrated in this report leads naturally to speculation regarding the future applications of the technique in areas other than the traditional bench test or trainer role, in which real-time simulation is generally cast. In particular, modern control systems rely, to some extent, on a mathematical or numerical representation of the plant to perform both fault detection and resource management functions. Currently, the microprocessor hardware utilized in these systems can support only rudimentary mathematical models. These are generally manifested as tables of numberical data and simple linear dynamic representations.

As the cost of microprocessors decreases, greater capability will become available. The techniques demonstrated in this project offer one approach to achieving the greatest cost/performance benefit possible from these devices. By distributing the mathematical model among several low-cost microprocessors, the greater potential of nonlinear modelling could be utilized in fault detection and in both steady-state and transient performance scheduling tasks. Fault detection by the model would be achieved by comparing sensed values from the plant to those of the model. On a short-term basis, failed control inputs would be eliminated from consideration through this process. At the same time, deterioration in the plant hardware would be determined from similar comparisons and used to correct the model. At any point, then, a best estimate of the current status of the plant would be available. The model could then be utilized to perform steady-state and transient scheduling tasks, without relying directly on measured plant parameters. Real-time nonlinear modelling would also be useful in plant health analysis applications. The modelling aspects, including the parallel implementation, are quite similar to that required for a model-based control mode. It is likely that the first use of these techniques will be in this sort of noncritical role.

IX. REFERENCES

1. McLaughlin, P., "A Technique for the Implementation of Nonlinear Models as Real-Time Digital Simulations", Proceedings of the 1980 Summer Computer Simulation Conference, Seattle, Washington, August, 1980.

2. Blech, R.A., and Arpisi, D.J., "An Approach to Real-Time Simulation Using Parallel Processing", Proceedings of the 1981 Summer Computer Simulation Conference, Washington, D.C., July 1981.

3. McLaughlin, P., "The Parallel Implementation of a Nonlinear Real-Time Simulation Technique", Proceedings of the 1983 Summer Computer Simulation Conference, Vancouver, B.C., July 1983.

4. Daniele, C.J. and McLaughlin, P., "The Real-Time Performance of a Parallel Nonlinear Simulation Technique Applied to a Turbofan Engine," Proceedings of the 1984 SCS Multiconference, San Diego, California, February 1984.

5. Gear, C.W., "Numerical Initial Value Problems in Ordinary Differential Equations", Prentice Hall, 1971.

6. Broyden, C.G., "Quasi-Newton Methods and Their Application to Function Minimization", Mathematics of Computation, 21, pp 568-581.

7. MC68000 16-Bit Microprocessor User's Manual, MC68000UM (AD2), Second Edition, Motorola, inc., Jan. 1980 (Preliminary).

APPENDIX

THE REAL-TIME PERFORMANCE OF A PARALLEL, NONLINEAR SIMULATION TECHNIQUE APPLIED TO A TURBOFAN ENGINE*

Carl J. Daniele
NASA-Lewis Research Center
Cleveland, OH

Peter W. McLaughlin
Pratt & Whitney Aircraft
East Hartford, CT

ABSTRACT

Under contract to NASA-Lewis Research Center, an investigation into the real-time capability of the parallel version of a nonlinear technique developed earlier was undertaken. The technique was applied to a typical turbofan engine model, similar to those used in conventional non-real-time practice. A serial FORTRAN program was written which emulates a variety of possible parallel processor architectures, including the Real-Time Multi-Processor System being developed by NASA-Lewis Research Center. This program has been utilized to evaluate the technique in its parallel implementation and to predict speed-up factors that would be expected in practice. Results which demonstrate the real-time performance of the technique as a function of the number of processors utilized are presented. The accuracy and stability of the technique as a function of the update rate and arithmetic precision is also documented.

INTRODUCTION

This paper presents work accomplished under NASA Contract NAS3-23283, "Parallel Processor Engine Model", which was conducted for NASA-Lewis Research Center in support of their Real-Time Multi-Processor System. The principal objective of this effort was to demonstrate the capability of the parallel version of a nonlinear, real-time simulation technique developed earlier (Reference 1). This technique, which is based on those used in non-real-time dynamic simulation, is well suited for parallel implementation.

This technique readily accommodates stiff models and provides stable operation at time increments larger than conventional explicit methods. However, its computational requirements represent a significantly greater cost than incurred by other approaches based on linearized models. Therefore, its widespread use has awaited expected advances in digital processing technology.

The results obtained in the contracted effort were directed primarily at the processor configuration specified by NASA/LeRC (Reference 2) but also extend to parallel concepts that lie beyond the capability intended for that device. Using this capability, a wide variety of parallel implementations of the model and the real-time technique have been evaluated in the course of the performance of the contract. The intent of these studies is to determine the most cost-effective arrangement of the model and the real-time technique for a given parallel processor architecture.

*This work was conducted under contract NAS3-23283 for NASA-Lewis Research Center.

REAL-TIME SIMULATION UTILIZING PARALLEL PROCESSORS

Recent experience with real-time simulations executed in support of closed-loop bench control system development testing has served to reinforce our belief in this practice. The models used in these tasks are based on a piece-wise linear approach, however, and suffer from accuracy problems unless the models are updated frequently. This is a time-consuming and expensive process. Linear models also tend to be less flexible in modelling the effects of secondary control systems. Both of these problems could be accommodated by the use of nonlinear models.

Parallel processing offers an alternative approach to real-time turbofan engine simulation. Instead of improving update rates by the acquisition of faster serial processors, slower, less expensive processors operate on different parts of the computing task, thereby achieving a reduction in the time each of them consumes. The effective update rate of this architecture is the longest of the individual execution times.

The next generation of real-time turbofan simulations will be required to execute models consistent with the requirements of multi-loop control systems at update rates several times greater than current practice. At the same time, market pressures will force the utilization of the most cost-effective approaches available. It is quite possible that arrays of low-cost microprocessors may offer an economical means of achieving nonlinear real-time simulation capability.

DESCRIPTION OF THE ENGINE MODEL

The performance aspects of the model used in this program are defined in a conventional manner. Component representations consist of combinations of algebraic and tabular data. Pressures, temperatures, and gas flows describe the conditions at the interfaces between components. The model is defined in a completely implicit fashion.

In order to provide the most stringent test of the parallel technique possible, especially with regard to precision, it was decided to include fluid continuity effects. This provided time constants which ranged from .00025 seconds up to 10 seconds for transient heat transfer phenomena. With time increments expected to be on the order of 10 to 30 milliseconds, the stability of the technique is well tested when applied to this model.

The model requires 18 iteration variables in its evaluation. Twelve of these are state variables representing rotor speeds, metal temperatures, pressures, and actuator outputs. The remaining six are required by the component performance representations.

ACCOMMODATION OF STIFF ELEMENTS IN THE MODEL

Gear's integration formula is applied to those states whose time constant is large compared to the time increment. For the stiff states associated with continuity effects, however the use of the integral form results in ill-conditioned Jacobian matrices. Therefore, a differential form is chosen which, at very large time increments (relative to the time constant), essentially negates the dynamic effect and degenerates to a simple iterative relationship.

Experience with this approach has been quite good. Stable responses are obtained with 15-bit precision at time increments varying from 1 up to 35 milliseconds. While current applications are well served by real-time simulation update rates of 10 to 20 milliseconds, future models will be required with response capability an order of magnitude greater. Proof of the technique in the higher frequency regime is an important consideration in its evaluation.

PARALLEL TECHNIQUE

Reference 1 details the development of the serial version of the technique. It is a truncated version of an implicit technique used in conventional non-real-time dynamic simulation. In its real-time manifestation, the technique demonstrates excellent stability on very stiff models; but, since it is an explicit method, it is not A-stable. In the range of update rates experienced in current practice, however, it provides evaluations of consistently defined turbofan engine models with very high reliability.

The technique is a quasi-Newton nonlinear solver, where the inverse Jacobian is updated by Broyden's method. Gear's stable second-order integration formula reduces spurious response to negligible levels. The nominal mode for this method is one in which iteration errors are reduced successively until all lie within a given tolerance band. This is termed a fully converged emulation. In the real-time version of the technique, each execution (corresponding to a given time point) is treated as an iteration attempting to converge the errors. With an appropriate modification to the Broyden update, the method reduces to a variety of predictor corrector, where predicted values are used to evaluate the model and are then corrected by the inverted Jacobian obtained from Broyden. The corrected values form the basis for the next prediction and are output to the control system hardware. The Broyden algorithm utilizes changes in both iteration variables and errors, as well as the current inverse Jacobian, to compute the changes to that matrix required to attain convergence.

Although the algorithm appears computationally complex, the communication of data between the different elements is quite manageable. This, coupled with the flexibility afforded by an implicit model formulation, makes the technique an attractive candidate for parallelization. In this discussion, it will be assumed that the transfer of data between parallel processors can only occur once per time step.

For a given arrangement of model segments on the available processors, a set of iteration variables necessary to solve the set of distributed equations in a simultaneous fashion can be identified. Each processor must be capable of calculating updated values of the iteration variables required on that processor, and it must compute errors associated with the values of iteration variables calculated in the model. This specifies which elements of the vectors involved are to be computed on a given processor.

Figure 1 shows the inter- and intra-processor data flows required by the real-time technique. This diagram also shows the arrangement of components of the technique on one of the parallel processors. The solid lines lying within the boundaries of the processor indicate data flow between elements on that processor. The double lines show data that must be transferred to other processors and be received from them. In the arrangement shown, the operations performed in Broyden depend on the distribution of iteration variables among the processors.

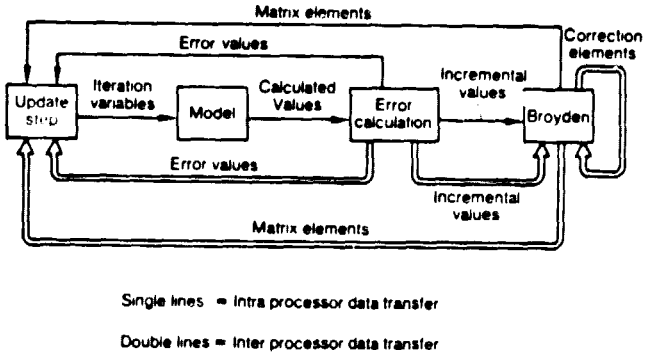


Figure 1 Data Flow Requirements; Coresident Broyden Algorithm

While this technique provides improved execution rates, it suffers from a major deficiency. As more processors are added and the model is distributed further, greater numbers of iteration variables are required. This leads to timing penalties that overwhelm the improvement derived from parallelization. This appears to be a fundamental limitation of synchronous data transfer applied to this technique.

An alternative approach was devised where the Broyden algorithm could be executed simultaneously with the remainder of the simulation. At least half the processors are used for Broyden, which reduces the number of iteration variables required in a given processor configuration. Since the allocation of Broyden computations is no longer connected to the model arrangement, it can be distributed in such a way as to balance the computing load among the processors.

Figure 2 shows the arrangement of two processors where the Broyden algorithm is executed simultaneously with the remainder of the technique.

This arrangement proved to be capable of achieving adequate real-time performance applied to the target system specified by NASA-LeRC; namely, eight to ten MC68000 processors serviced by synchronous data transfer. Data presented later in this paper documents the results obtained.

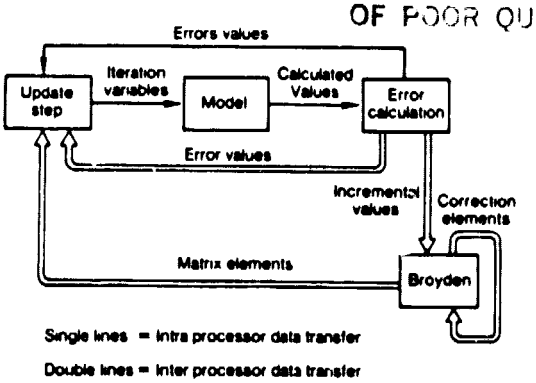


Figure 2 Data Flow Requirements; Simultaneous Broyden Algorithm

ASYNCHRONOUS DATA TRANSFER

An asynchronous data transfer approach has proven to result in substantial execution time improvements over the synchronous scheme originally treated. This method utilizes "crosstalk" between the processors to transfer data resulting from model calculations on one processor to another where that data is required for other calculations. This approach represents an alternative to the use of iteration variables as a means of communicating this data.

In this scheme, only the basic set of iteration variables required for the sequential model are allocated. It is assumed that the iteration variable update step will be synchronized so that all the model executions begin simultaneously. This ensures that the results of the iteration variable update will be available for "crosstalking" to processors other than the one where the update takes place.

Figure 3 shows the schematic arrangement of segments in an asynchronous crosstalk scheme. The segment on the receiving processor must be delayed by an amount which is the difference between the ending time of the sending segment and the beginning time of the receiving segment.

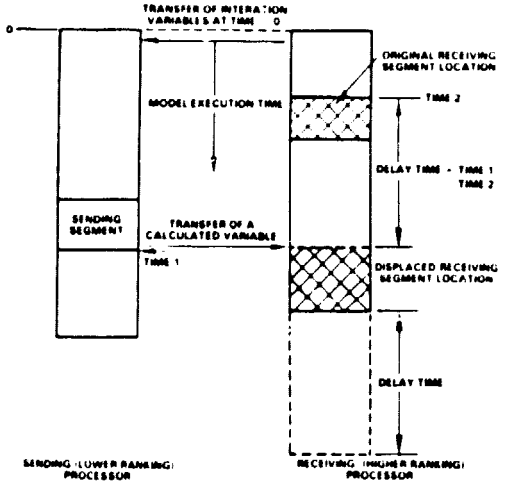


Figure 3 Illustration of Asynchronous Data Transfer Crosstalk

In order to prevent contention between processors for precedence in the crosstalk hierarchy, the processors have been ranked in ascending order of their identification number. Crosstalk is only allowed to occur from lower-numbered to higher-numbered processors. Since this is the direction in which the default segment sequences are allocated, there is no requirement for crosstalk in the opposite direction from that preferred. For other segment arrangements, variables that must be crosstalked "upstream" are identified as iteration variables, rather than crosstalk variables, and are treated in the same way as the basic set of iteration variables associated with the sequential model.

CONFIGURATION SURVEY

It was recognized early in this work that the use of a detailed emulation of the parallel processing systems under consideration would be invaluable in examining all the possible alternative approaches. This FORTRAN program, executing serially, allows the user to evaluate a wide range of parallel arrangements. In addition, the accuracy and stability of the technique, operating with identical precision as the target processor, can be directly evaluated. Based on the timing data collected from the model and the evaluation routines, and the arrangement of simulation functions chosen by the user, a time increment requirement is computed as the sum of the largest execution time of the processors plus the time required to transfer data among the processors. A large matrix of configurations was executed using the NASA/LeRC Real-Time Multi-Processor Simulator as the target. In these runs, the model was distributed among one to eight processors. Four variations of the parallel technique were executed in both the synchronous and the asynchronous data transfer modes. This resulted in a total of 64 different configurations being evaluated. The synchronous data transfer mode which produced the minimum time increment, and which lies within the design parameters of the RTDS, utilizes five processors. The model is distributed between two processors. The simultaneous Broyden function is implemented on the remaining three processors. The time increment for this configuration is .0244 seconds. This represents a reduction factor of 54 percent, based on the one-processor requirement of .0536 seconds. Figure 4 tabulates the results obtained from this study.

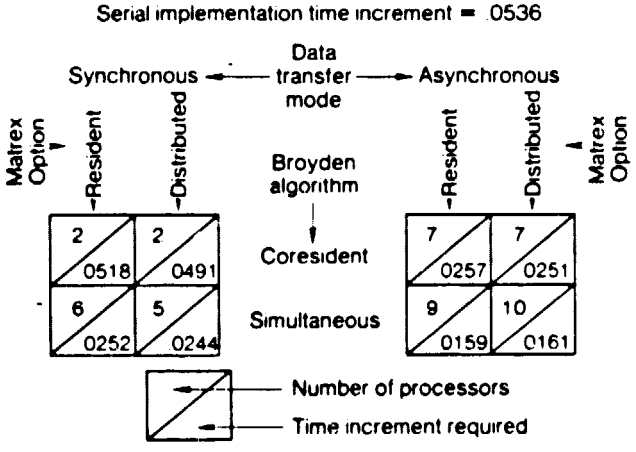


Figure 4 Configuration Survey Summary

As can be seen from these results, the use of asynchronous data transfer offers a significant improvement in the update rate possible with this technique. A minimum value of .0159 seconds was obtained for an asynchronous configuration consisting of nine processors.

ACCURACY

The real-time technique is explicit, and hence, manifests errors that are eliminated in an implicit, fully converged implementation. It is appropriate to measure this effect in terms of the size of these errors experienced by the model during simulation operation. In order to provide a single figure-of-merit, an average RMS error is computed from a summation covering the entire test transient. The errors are relatively small and well within the accuracy requirements of the application.

Fifteen-bit arithmetic precision was assumed for the cases discussed above. Greater precision is more costly in terms of update rate, so that the capability of the technique to tolerate relatively low-precision operations is an important characteristic recommending its use. Figure 5 shows the effects of varying arithmetic precision. Twenty-four-bit precision represents typical 32-bit floating point precision, while 30-bit precision is afforded by the use of scaled 32-bit integer operations. While a loss in relative accuracy is evident as precision is decreased, the absolute level obtained at fifteen bits is more than adequate for the intended applications.

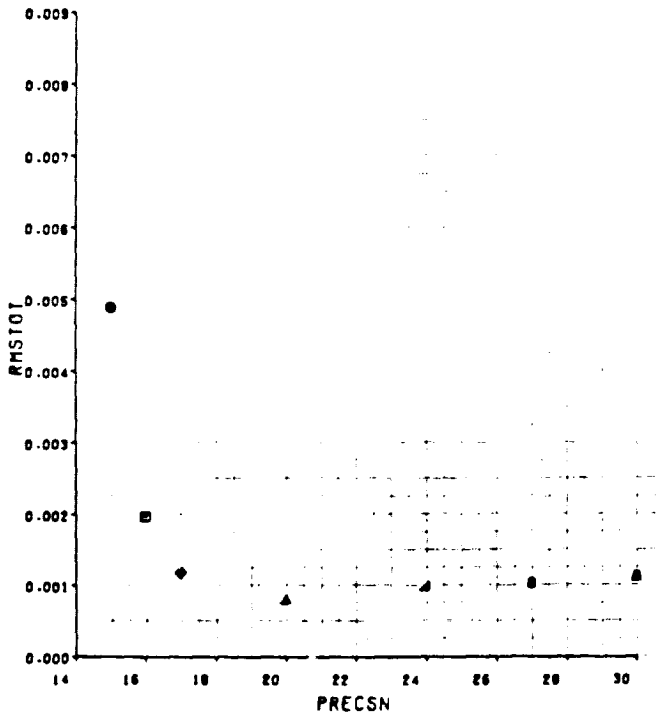


Figure 5 Merit as a Function of Precision

STABILITY

The use of 15-bit precision tends to reduce the stability margin afforded by the use of 32-bit data representations. The characteristic failure mode associated with the technique is one in which nominally stable operation becomes unbounded in the vicinity of certain model inconsistencies. This property of the technique is related primarily to the Broyden algorithm. Under the influence of models that may be locally ill-conditioned and with limited precision available in real-time, the ratio of increments used to update the matrix may become indeterminate to a degree that causes a breakdown in the process. This can be prevented by establishing a lower limit on the size of the denominator factor used in the matrix update. However, if this tolerance is set too large, the algorithm will fail to provide updated information to the matrix. In this event, model nonlinearities will eventually cause failures similar to those discussed above.

DYNAMIC CHARACTERISTICS OF THE TECHNIQUE

The accuracy and stability considerations discussed in the preceding sections certainly affect the operation of the real-time simulation in a closed-loop environment where it interfaces with hardware systems. However, the dynamic characteristics of the real-time simulation itself, viewed as an element of a closed-loop test bench, are of equal importance. The most significant characteristic of digital real-time simulation is the time delay associated with the discrete process involved. While relatively low-frequency model inputs experience little distortion due to the delay, frequencies approaching the simulation update rate generate considerable spurious responses. If the response of the system is substantially affected, erroneous tests can be executed--and, at worst, unstable system response results.

Generally, the delayed signals produced by the simulation are compensated by extrapolating their values ahead in time somewhat before being transmitted to the system. Predictive compensation does not eliminate the problem but does improve the dynamic accuracy of the model insofar as the real system is concerned.

In the synchronous parallel version of the real-time technique reported here, the delay imposed is increased to two full cycles due to the constraint that data is only transferred once per cycle. While appropriate extra compensation will alleviate this problem at lower frequencies, it is probable that the stability characteristics of closed-loop systems employing this technique in its parallel implementation will not be improved, even though higher update rates are achieved. In any event, stability should not be a constraining factor as long as maximum update rates are correctly specified in the design of the closed-loop facility.

The use of asynchronous data transfer allows a return to the original, serial delay effect. This property, coupled with the much higher update rates resulting from the use of more processors, promotes this approach as a means of achieving extremely high update rates.

CONCLUSIONS

1) Parallel microprocessors employing synchronous data transfer are capable of providing real-time simulation capability for turbofan engine applications.

2) The real-time simulation technique utilized is capable of stable operation when implemented on parallel processors.

3) Asynchronous data transfer greatly enhances the real-time performance of the parallel technique.

4) Fifteen-bit scaled-integer arithmetic provides adequate precision for the technique if the nonlinear model is consistently well defined.

5) The simulation technique utilized here in either its real-time or fully converged modes can be successfully parallelized to any degree desired if high-speed asynchronous data transfer between the processors is employed.

REFERENCES

1. McLaughlin. P., "A Technique for the Implementation of Nonlinear Models as Real-Time Digital Simulations", Proceedings of the 1980 Summer Computer Simulation Conference, Seattle, Washington, August, 1980.

2. Blech, R.A. and Arpisi, D.J., "An Approach to Real-Time Simulation Using Parallel Processing", Proceedings of the 1981 Summer Computer Simulation Conference, Washington, D.C., July, 1981.

3. McLaughlin, P., "The Parallel Implementation of a Nonlinear Real-Time Simulation Technique", Proceedings of the 1983 Summer Computer Simulation Conference, Vancouver, B.C., July, 1983.