Time-Accurate Turbine Engine Simulation in a Parallel Computing Environment Part II - Software Alpha Test[1]

M. A. Chappell and B. K. Feather

Sverdrup Technology, Inc., AEDC Group
Arnold Engineering Development Center, TN

Abstract. The ability to rapidly execute a high-fidelity turbine engine simulation is essential in automating data validation processes during developmental engine testing. Further, execution of a simulation in real time enables validation of a continuously varying data stream concurrently with the data acquisition task.

Component-level turbine engine simulations provide high-fidelity steady-state and time-accurate transient engine performance computations but are not typically applied in a real-time environment. This paper presents an approach for distributing a component-level simulation task in a parallel computing environment in an effort to achieve real-time operation.

A turbine engine simulation was restructured to operate in a parallel computing environment and tested to quantify the resulting impact on simulation fidelity and execution time. A software acceptance test confirmed the viability of the parallel simulation approach and guided the formulation of refinements. The refinements were incorporated, and the program was reevaluated in terms of repeatability, speedup, and scalability. The approach, software alpha test results, and direction of future work are summarized herein.

Introduction

Turbine engine testing at the Arnold Engineering Development Center (AEDC) is conducted to evaluate engine operation over a wide variety of power conditions and simulated altitude conditions. Thousands of sensors, many producing measurements at rates in excess of one hundred samples per second, are typically installed in the engine and in the test facility to measure aerothermodynamic performance. Consequently, a typical 8-hour test can produce three billion samples of aerothermodynamic performance data. The challenge is to ensure the validity of the data, monitor the condition of the engine, and promptly identify anomalies.

The countless variations of steady-state and transient engine operation and the necessity to distinguish between sensor anomalies and abnormal engine deterioration, combined with the large volume of data, overwhelm the capabilities of traditional data

[1] The research reported herein was performed by the Arnold Engineering Development Center (AEDC), Air Force Material Command. Work and analysis for this research were performed by personnel of Sverdrup Technology, Inc., AEDC Group, technical services contractor for AEDC. Further reproduction is authorized to satisfy needs of the U. S. Government.

V.N. Alexandrov et al. (Eds.): ICCS 2001, LNCS 2073, pp. 1237–1250, 2001.

validation methods. Although traditional methods produce meaningful results, they are labor-intensive and time-consuming. Consequently, application of traditional methods is typically restricted to a fraction of the available data, thus diminishing the ability to detect anomalous data and intermittent events.

An approach was developed for a fast, comprehensive, and automated data validation process. The process consists of several analysis and modeling methods which together provide a real-time, model-based data validation tool. The methods include an event detection system, a rule-based expert system with more than 150 checks, and an engine model-based fault detection and diagnostic system (Fig. 1). The engine model-based element of the system relies on a component-level turbine engine model (CLM) which employs fundamental physical laws to relate measurements to each other. The CLM is an industry-accepted method [1-3] for simulating the aerothermodynamic performance of turbine engines but is not typically used when real-time simulation is required. However, the CLM provides the level of accuracy and detail required for data validation and fault diagnosis. Other turbine engine modeling methods are available but may compromise accuracy and detail in order to increase execution speed [4]. Furthermore, the CLM is more easily adapted to the frequent changes that are expected during engine development.

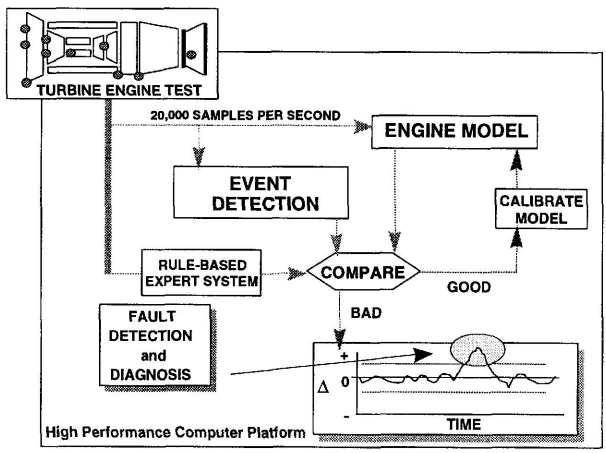

Fig. 1. Real-Time, Model-Based Data Validation System

The CLM must process each data sample at speeds which match or exceed the data-sampling rate since the CLM is only one element of an automated process which monitors a data stream whose measurements vary continuously. Otherwise, a backlog of data samples will grow while waiting for earlier \!data samples to be processed. A "real-time" processing rate is defined as a rate that matches the sampling rate. A requirement also exists for a rate which is faster than the "real-time" rate. This additional requirement addresses a need to process recorded data in a time span which is shorter than the duration of the test. For example, a four-hour data tape, off-loaded from a test vehicle, could be processed in less than four hours.

The challenge is to implement a computing strategy that enables the CLM to execute in real time or faster. Parallel computing was selected as a strategy to provide real-time execution. This strategy also can be scaled up, allowing extension to offline "faster than real time" applications. A temporal decomposition of a time-accurate turbine engine CLM is proposed in which the CLM is replicated on multiple computer processors (CPU) within a high-performance multiple-CPU computer. Each CPU processes a set of synchronized data samples as other sets are processed on additional CPU's. Temporal decomposition was selected rather than spatial decomposition, which is often used for computational fluid dynamic (CFD) applications [5] because previous CLM work suggests that spatial decomposition of a CLM is ineffective [6,7]. Additionally, temporal decomposition of a CLM is more easily distributed to an increasing number of CPU's. Each level of software testing imposes more rigorous standards than the previous, starting with SAT as an entry level assessment of the software and progressing through alpha testing and beta testing.

The objective of this work is to provide an automated, real-time, model-based test data validation computer code. The CLM is one element of an approach to permit comprehensive validation of test data. Each element, including the CLM, is subject to standard software development practices including Software Acceptance Testing (SAT), Alpha testing, and Beta testing prior to a production release.

The subject of this paper is the parallel implementation strategy for the time-accurate CLM. Results of software alpha testing (α-test) and recommended refinements to the strategy are also discussed.

Component-Level Model Description

The component-level turbine engine model (CLM) provides a lumped-volume, component-level, time-accurate simulation applicable to arbitrary engine designs. The CLM is capable of simulating off-operating-line engine operation and uses widely accepted component-matching principles [1-3]. An engine control simulation may also be included as an additional program module.

The CLM combines physical relationships that govern engine operation with empirical relationships that describe individual component performance. The result is an adaptable model for which the effects of changes to engine attributes (e.g., components, configuration, and controls) are incorporated by making changes to the corresponding model attributes (i.e., components, configuration, and controls modules). In addition, the component-matching approach quantifies the effects of changes to the engine attributes, enabling a prediction capability for the data validation and fault diagnosis process.

The CLM is an assembly of components constrained to operate in unison to simulate the engine. An augmented turbofan engine, for example, may include a variable-geometry fan and compressor, combustor, high-pressure and low-pressure turbines, fan bypass duct, mixer, augmentor, and variable exhaust nozzle (Fig. 2). The component models combine thermodynamic process equations with empirically determined component performance relationships to simulate component

performance. An iterative technique is used to satisfy the implicit relationships that constrain the assembly to mass, momentum, and energy conservation principles. Measured engine control variables are used to govern model operation. The effects of rotor acceleration, heat transfer, and off-schedule variable geometry are included, providing a simulation of steady-state and transient engine operation ranging from engine starting conditions to maximum power. The CLM provides accurate simulation of operating temperatures, pressures, mass flows, and rotational speeds for each of the components illustrated in Fig. 2.

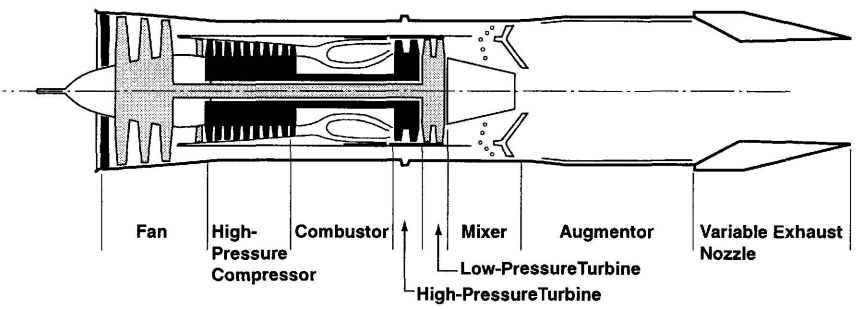

Fig. 2. Augmented Turbofan Engine

Figure 3 illustrates the high level of agreement achievable for model calculations as compared to test measurements for a turbine engine propulsion system. Typical parameters computed by the model include compressor speed and combustor pressure. Agreement briefly exceeds 11 percent but generally is within 5 percent during an in-flight restart sequence, in which the engine experiences maximum power, shutdown, starting, and idle operation.

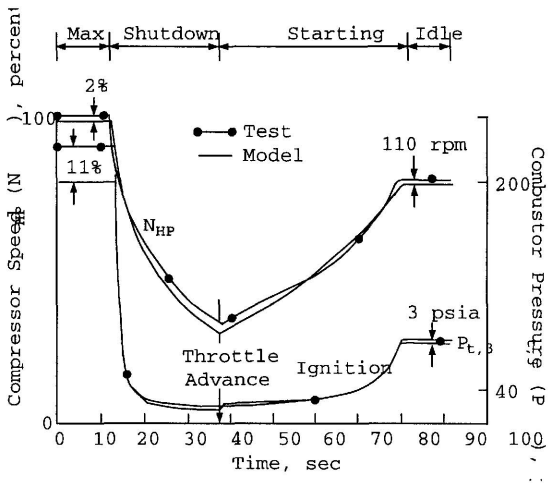

Fig.3. Comparison of Model Calculations to Test Measurements for an Augmented Turbofan Engine

The primary time-dependent effect for the time-accurate CLM is the influence of rotor inertia on engine operation. Heat transfer between the gas path and the engine hardware is a significant but secondary effect. The physical effects of rotor inertia and heat transfer on engine operation are dependent on the time-related history of the affected parameters, and consequently the simulation of these effects is dependent on previously computed values of the affected parameters. The dependence on previous values of the affected parameters imposes a serial character on the simulation computations. The serial character of the physical phenomena and their simulation, coupled with the serial character of numerical integration techniques employed by the CLM, presents a challenge to the parallel computing strategy for the CLM.

CLM Operation in a Test Environment

The CLM, as used in a test environment, is implemented without an engine control simulation module. Avoiding a requirement for a control simulation module eliminates a potential source of error in the data validation process and simplifies the CLM. In the absence of a control simulation, selected test measurements are used as inputs to the CLM to prescribe its forcing functions.

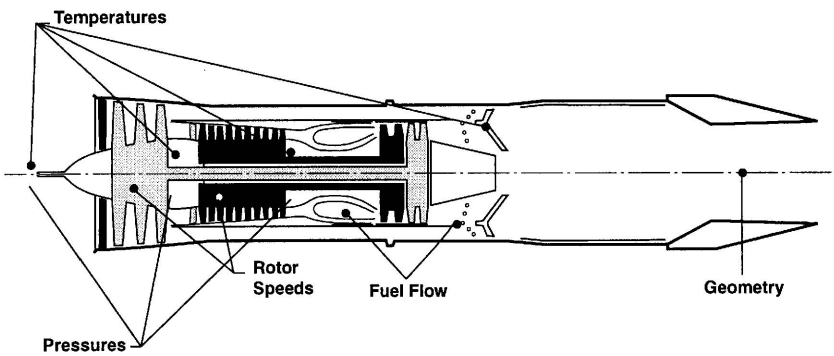

Fig. 4. Test Measurements Used as Inputs to CLM

Typically measurements of the controlled variables, such as fuel flow, are used as inputs along with measurements of the environmental parameters, such as inlet pressure and temperature. Figure 4 illustrates a typical set of pressure, temperature, speed, flow, and variable geometry measurements used as model inputs. Rotor acceleration computed from a time history of rotor speed measurements is also input to the model together with the speed measurement.

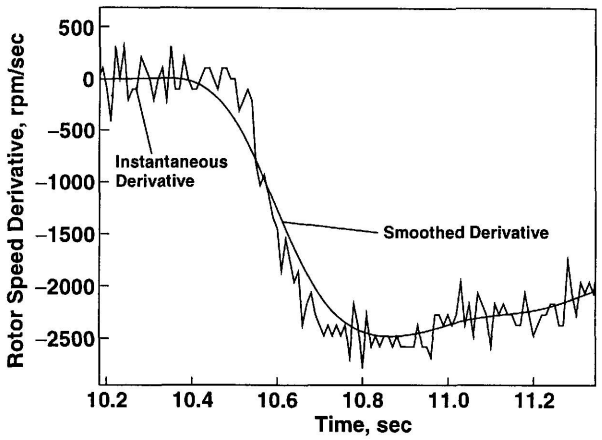

Fig. 5. Engine Deceleration: Rotor Speed Derivative as a Runction of Time

Time-dependent fluctuations in signals for a particular measurement are normal. Although normal, these variations are detrimental to the numerical stability of the model since the model is not designed to respond to rapid (>20 Hz) variations of operating conditions. Unstable model operation reduces fault detection and diagnosis effectiveness and leads to lengthy execution times; therefore, the measurements are smoothed using a time-averaging technique to minimize measurement noise while preserving the mean value of the measurement. Typical variations among fan speed measurements are shown in Figure 5 along with smoothed values used as input to the model.

Parallel CLM Implementation

The time-decomposition of the time-accurate CLM enables a replicated worker approach for a parallel computing implementation of the CLM. The CLM code is replicated on each CPU of a parallel high-performance computer (HPC). Although the results shown herein are from shared memory HPCs, the parallel implementation is also achievable on distributed architectures. All of the worker processors are controlled by a single master processor (Fig. 6). The master processor transmits a user-determined number of data samples to each worker, and the worker processes the samples with its own replication of the model. The alpha version (a-version) of the code ensures that each replicated worker processes a user-selected interval of contiguous samples in contrast to the previous version [8], which processed noncontiguous samples (i.e., each successive sample was processed by a different CPU). The replicated workers receive another interval of samples upon completing each interval; however, the ensuing interval is not contiguous with the previous interval. A coarse-grain worker, discussed below, provides the starting time-dependent boundary conditions for each interval. Additionally, to improve

computational efficiency, all data are loaded into memory before processing is initiated. This sequence precludes disc contention bottlenecks.

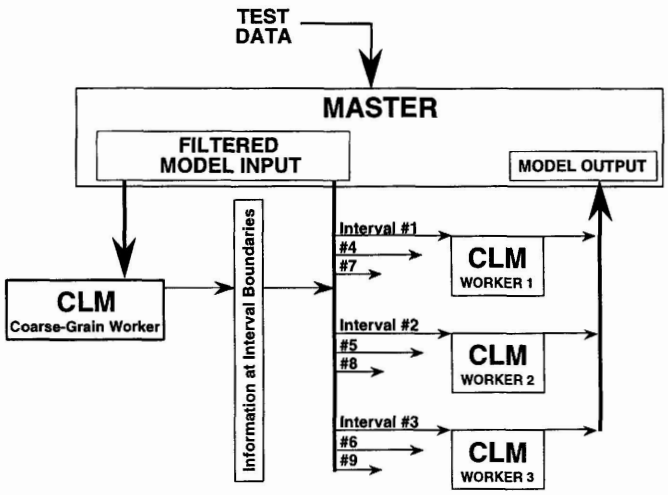

Fig.6. Replicated-Worker Approach for Time-Decomposition of CLM
(Three Replicated Workers)

This approach, in which each worker processes an interval of samples, promotes model convergence, reduces the interprocess communication burden, and introduces a challenge in addressing time-dependent effects. Model convergence improves because the model is cognizant of the effect of neighboring samples on the computations. The communication burden is reduced because transmitting many samples with one communication instruction is more effective for the worker than transmitting the same samples with an individual instruction for each sample. The challenge for each worker is to compute the time-dependent effects (e.g., rotor acceleration and heat transfer [8] between nonadjacent intervals. Thus a coarse-grain worker was designed to address the challenge.

The coarse-grain worker accesses the same data samples as the replicated workers but processes only a fraction of them. Also, the coarse-grain worker processes the samples contiguously and computes the time-dependent effects and corresponding boundary conditions for the replicated workers (Fig. 7.). The coarse-grain worker completes its task before the replicated workers need the time-dependent information because the coarse-grain worker processes fewer samples. The coarse-grain worker provides the time-dependent effects to the replicated workers for the starting time of each interval, and the result is fast and accurate computation from the replicated workers.

However, a balance between the granularity of the coarse worker (i.e., the fraction of samples processed) and the scalability of the approach (i.e., the number of replicated workers) is required. As more replicated workers are added to the configuration, the

aggregate rate at which they collectively process the intervals eventually exceeds the rate at which the coarse-grain worker computes the boundary conditions. Further

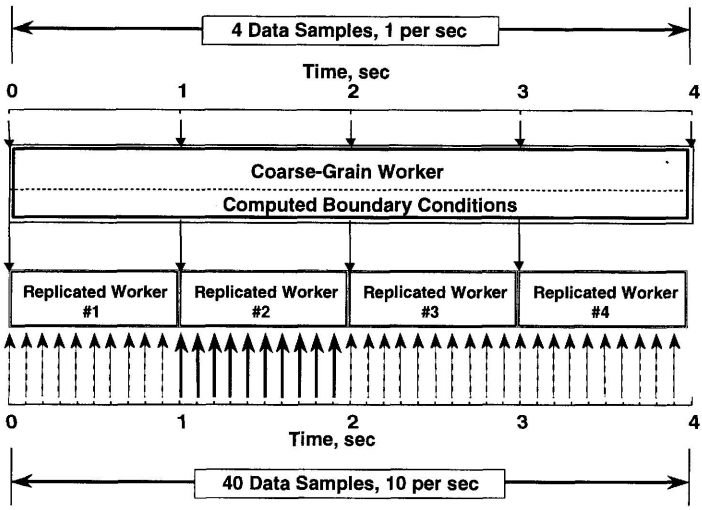

Fig. 7. Example of Relationship Between Coarse-Grain Worker, Replicated Worker, and Data Samples

reduction of the fraction of samples processed by the coarse-grain worker increases its rate but eventually compromises the fidelity of the calculations. Results of the α-test indicate that an acceptable balance is achievable for parallel configurations of up to 32 processors. Parallel configurations of larger than 32 processors have not been tested.

The α-implementation also included a feature to operate the CLM code in a batch mode, thus requiring no interaction from the user. The batch mode enabled extensive automation of software testing. Job scheduling queues on the HPC platforms automatically processed all of the software test cases, testing each case with a wide variety of parallel configurations. Operating within the queuing environment also ensured a stable and consistent environment to conduct repeatable execution time measurements.

Software Alpha-Testing

Two aspects of α-testing were crucial: repeatability verification on multiple CPU configurations, and execution speed measurements. Repeatability verification ensures that the results obtained from a variety of parallel configurations are the same within a tolerance. The execution speed measurements provide an indication of scalability, which is defined as the effectiveness of adding additional CPUs to the process either to reduce execution time of the CLM or to process larger problems.

Repeatability verification, described as the replication of results within a tolerance, was required for all multiple-CPU configurations. Differences in the results between the serial execution (single CPU) and multiple-CPU configurations are attributed primarily to a less accurate starting state for each multiple-CPU interval, a degeneration that results from the temporal decomposition. Acceptable tolerances for the differences were established for individual parameters and were based on the importance of the parameter in the fault detection process, the particular engine maneuver, and the convergence tolerance within the code. Typical comparisons between serial execution, which was considered the baseline, and multi-CPU configurations are shown in Figs. 8 through 13. Acceptable tolerances for α-testing are also contrasted to the higher SAT tolerances in the figures.

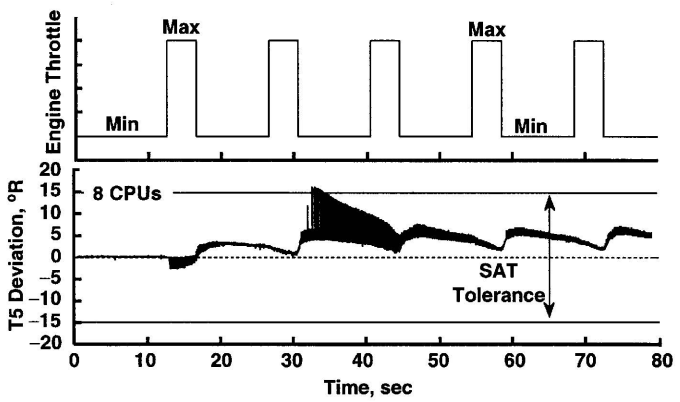

Fig. 8. SAT Turbine Exit Temperature Deviations with Additional CPU's

SAT results [8] are repeated (Figs. 8 and 9) for ease of comparison to α-testing results. Notable improvements in repeatability are apparent in Figs. 10 and 11 relative to SAT results. Figures 8 and 10 show the differences in turbine exit temperature between the serial configuration and an 8-CPU configuration for the SAT and α-implementations, respectively. Turbine exit temperature (T5) was selected for illustration because rotor acceleration and heat transfer affect it significantly. Similarly, Figs. 8 and 10 show the differences in turbine exit pressure between the serial configuration and an 8-CPU configuration for the SAT and α-implementations, respectively. Turbine exit pressure (P56) was selected for illustration because it is sensitive to model convergence tolerances. Time-dependent effects (e.g., rotor acceleration and heat transfer) are included in all cases shown.

The SAT results (Figs. 8 and 9) exhibit a deviation from the baseline during intervals in which the time-dependent effects are significant–in this case, during engine throttle excursions (also shown in Figs. 8 through 11). However, the α-implementation follows the baseline more exactly during the same intervals (Figs. 10 and 11), resulting in differences within one percent, thus verifying the improved repeatability of the α-implementation.

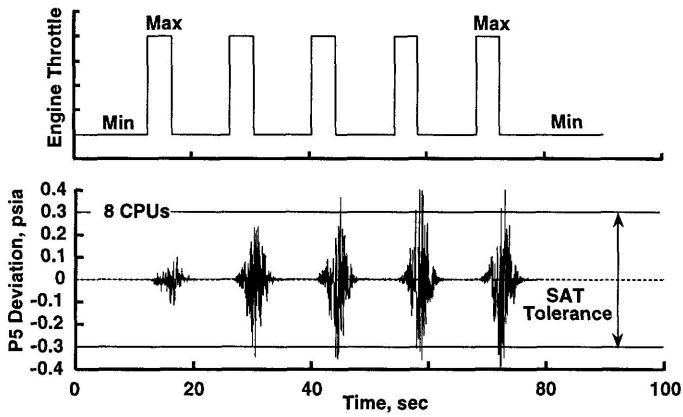

Fig. 9. SAT Turbine Exit Pressure Deviations with Additional CPU's

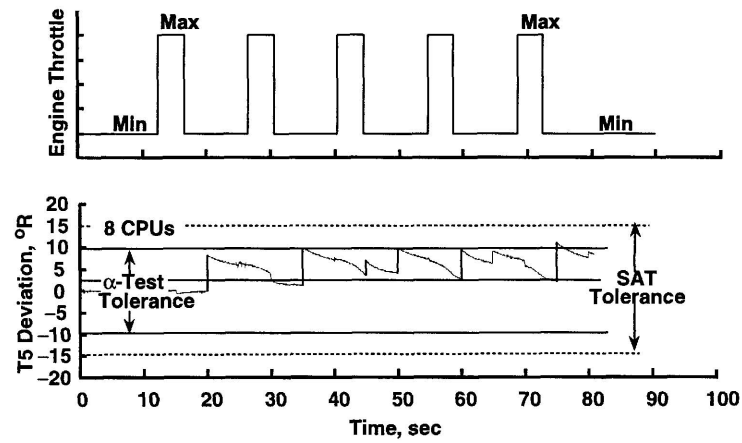

Fig. 10. V-Implementation: Turbine Exit Temperature Deviations with Additional CPU's

Execution speed was also measured during α-testing of the CLM. The same twelve distinct engine maneuvers used during SAT [8] were used during α-testing. Measurements were taken for serial executions of the CLM plus a variety of multiple-CPU configurations ranging from 4 to 32 CPUs on two different massively parallel, high-performance computers (SGI Origin 2000 and HP V2500). Representative results are shown in Figs. 12 and 13. Figure 12 illustrates the speed increase factor relative to the serial execution time (i.e., speedup) achieved by employing additional CPUs. Figure 13 illustrates the speed increase factor relative to the duration of the engine maneuver as an indication of "real-time" capability. Variation in speedup among different engine maneuvers results from the impact of the maneuver on the number of iterations required for model convergence (i.e., severe maneuvers require more iterations). Figure 12 indicates that the speedup of the α-implementation exceeds 50 percent of the maximum theoretical speedup up to the 32-CPU configuration.

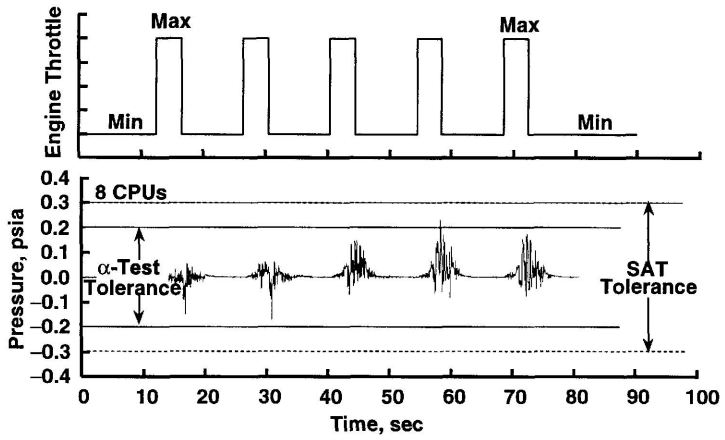

Fig. 11. ∇-Implementation: Turbine Exit Pressure Deviations with Additional CPU's

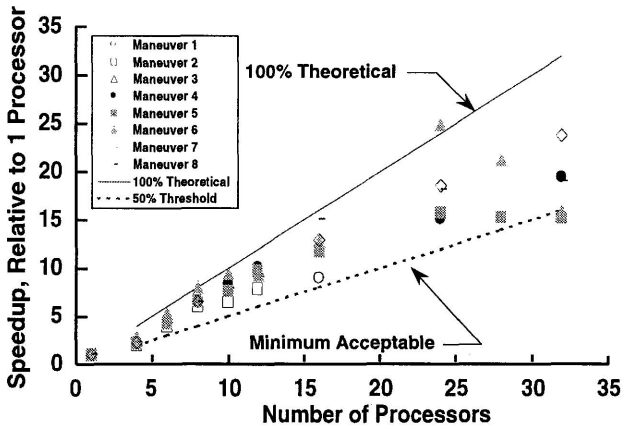

Fig. 12. ∇-Implementation: Parallel Processing Effectiveness in Terms of Speedup

Scalability of the α-implementation was also assessed. Scalability is quantified by the relationship between problem size, or throughput, and the number of CPUs. For the CLM code, scalability is the ability to process more samples within a prescribed length of time by employing more processors. As an example, to achieve 100-percent scalability for a problem size that is two times larger than a selected baseline problem, the code must achieve the same execution time as the baseline with twice as many processors. Longer execution times result in reduced scalability.

Scalability is measured in terms of an overall processing rate which is normalized by the number of CPUs to produce an average processing rate per CPU. Ideally, all parallel configurations will achieve the same normalized processing rate with problems that are sized proportionally with the number of CPUs. Less than ideal performance is realized for a variety of reasons including higher inter-processor communication activity. The normalized processing rate for a smaller problem is

selected as the baseline to which all other rates are compared as an indication of scalability.

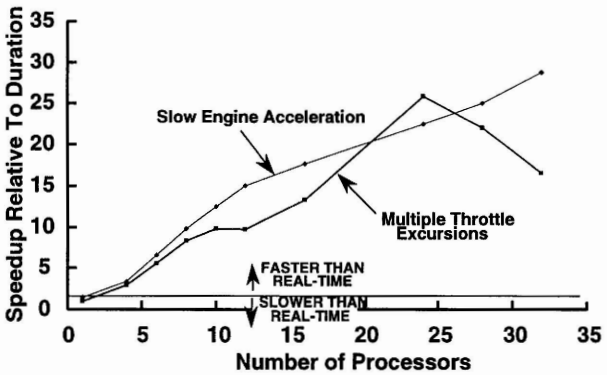

Fig. 13. ∇-Implementation: Speed Increase Factor Relative to Duration of Engine Maneuver

Figure 14 illustrates a comparison of scalability for a larger problem with many CPUs in contrast to a smaller problem with fewer CPUs. The larger problem is composed of four times more samples than the smaller but is otherwise equal in complexity. All values in Fig. 14 are referenced to the baseline case such that a rate equal to that of the baseline is considered ideal. High scalability (96 percent) is realized for the larger problem for 14 worker processors. Although the scalability diminishes as more CPUs are added, it remains above an acceptable level (50 percent) with up to 30 replicated workers.

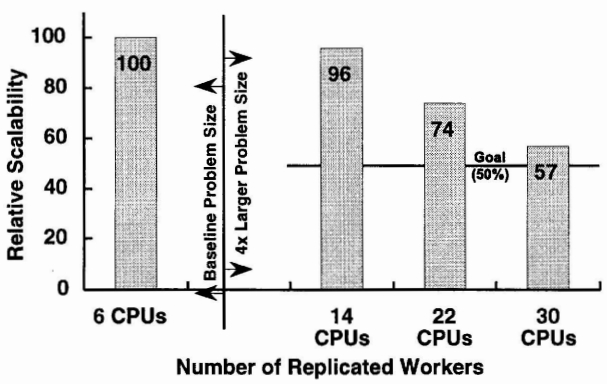

Fig. 14. ∇-Implementation: Relative Scalability
(Average Processing Rate per CPU Relative to Baseline)

Future Work

Three aspects of the code were identified as areas of improvement and the target of future work. The first area focuses on code modularity. Modularity is essential for application of the code in a variety of situations which include both off-line and on-line (i.e., real-time) operation and interactive use and batch operations. The second area focuses on generalization of the CLM code to facilitate application of the code to many different turbine engine designs. Finally, as processors and memory become faster and more expansive, incorporation of more advanced and computationally intensive models [9,10] becomes feasible.

Summary

Component-level turbine engine simulations provide high-fidelity, time-accurate, engine performance computations. An approach for distributing a component-level turbine engine simulation task in a parallel computing environment was implemented and subjected to software alpha-testing (α-test).

Software α-testing confirmed the viability of the component-level model (CLM) code in terms of repeatability verification, speedup, and scalability. Results of the a-testing indicate that the CLM can operate in real time and provide acceptable accuracy to serve as a critical element of a comprehensive, automated data validation process.

References

1. McKinney, J. S., "Simulation of Turbofan Engine, Part I: Description of Method and Balancing Technique," Air Force Aero-Propulsion Laboratory, AFAPC-TR-67-125-PT.-1, 1967.

2. Koenig, R. W. and Fishbach, L. H., "GENENG: A Program for Calculating Design and Off-Design Performance for Turbojet and Turbofan Engines," NASA TN D-6552, 1972.

3. Shapiro, S. R. and Caddy, M. J., "NEPCOMP: The Navy Engine Performance Program," American Society of Mechanical Engineers, 74-GT-83, March 1974.

4. *Real-Time Modeling Methods for Gas Turbine Engine Performance*, Aerospace Information Report, AIR4548, Society of Automotive Engineers, Warrendale, PA, December 1995.

5. Steger, J. L., et al., "A Chimera Grid Scheme," American Society of Mechanical Engineers, Proceedings of the Applied Mechanics, Bioengineering, and Fluids Engineering Conference, Houston, TX, June 20-22, 1983.

6. *Parallel Processor Engine Model Program*, NASA-CR-174641, January 1984.

7. Biegl, C., et al., "Automated, Real-Time Validation of Turbine Engine Test Data Using Explicit Parallelization on an Eight-Processor Pentium Platform," XIII International Symposium on Air-Breathing Engines, ISABE 97-7143, September 1997.

8. Chappell, M. A., et al., "Time-Accurate Turbine Engine Simulation in a Parallel Computing Environment, Part I: Software Acceptance Test," Second Annual International Test and Evaluation Association High Performance Computing Workshop, Edgewood, MD, June 1999.

9. Hale, A. A., et al., "Turbine Engine Analysis Compressor Code: TEACC -- Part II: Multi-Stage Compressors and Inlet Distortion," American Institute of Aeronautics and Astronautics, AIAA-99-3214, 1999.

10. Lytle, J. K., "The Numerical Propulsion System Simulation: A Multidisciplinary Design System for Aerospace Vehicles," XIV International Symposium on Air-Breathing Engines, ISABE 99-7111, September 1999.