ISABE 97-7143
Automated, Real-Time Validation of Turbine Engine Test Data Using Explicit Parallelization on an Eight-Processor Pentium® Platform

Csaba Biegl
Vanderbilt University
Department of Electrical Engineering
Nashville, TN
and
Donald J. Malloy and Mark A. Chappell
Sverdrup Technology, Inc., AEDC Group
Arnold Engineering Development Center
Arnold Air Force Base, Tennessee 37389

XIII INTERNATIONAL SYMPOSIUM
ON
AIR BREATHING ENGINES

September 7 -12, 1977
Chattanooga, Tennessee

AUTOMATED, REAL-TIME VALIDATION OF TURBINE ENGINE TEST DATA USING EXPLICIT PARALLELIZATION ON AN EIGHT-PROCESSOR PENTIUM® PLATFORM*

Csaba Biegl
Vanderbilt University
Department of Electrical Engineering
Nashville, TN

Donald J. Malloy and Mark A. Chappell
Sverdrup Technology, Inc., AEDC Group
Arnold Engineering Development Center
Arnold AFB, TN

ABSTRACT

A parallel distributed machine consisting of eight Pentium® PC clones is presented as a development environment that emulates a distributed workstation network. An overview of the model-based fault identification approach and typical test results are presented. Explicit parallelization on an eight-processor Pentium® platform is shown to provide a higher level of parallelization than with instruction-level parallelizing compilers. An interactive graphical user interface and tools for explicit parallelization of the recursive algorithms are described. The ease of implementation of the parallelization approach and highly satisfactory nature of the numerical results indicate the effectiveness of the distributed network for real-time data processing.

INTRODUCTION

Turbine engine testing at the Arnold Engineering Development Center (AEDC) is conducted to evaluate engine operation at a wide variety of simulated altitude conditions. Hundreds of sensors, each producing measurements at rates in excess of 100 samples/sec, are typically installed in the engine and test facility to measure aerothermodynamic performance. Consequently, a typical 8-hr test can produce 30 million samples of aerothermodynamic performance data. The challenge is to ensure the validity of the data, monitor the condition of the engine, and to identify anomalies promptly.

The countless variations of steady-state and transient engine operation and the necessity to delineate between sensor anomalies and abnormal engine deterioration, combined with the large volume of data, overwhelms the capabilities of traditional data validation methods. Although traditional methods produce meaningful results, they are labor-intensive and time-consuming. Consequently, application of the methods is typically restricted to a fraction of the available data, which diminishes the ability to detect anomalous data and intermittent events. In order to meet the requirements for a fast, thorough system, AEDC has undertaken a development program which uses a number of automated analysis tasks to maximize test efficiency. Automated analysis tasks include an event detection system, a rule-based expert system with more than 150 checks, and the model-based fault detection and diagnostic system (Fig. 1).

An automated approach that emulates the traditional data validation and engine condition monitoring processes is needed to ensure a comprehensive assessment. Nonlinear component-matching engine models embody the physical relationships employed in the data validation and engine condition monitoring processes and can provide a basis for automating them. However, in order to provide

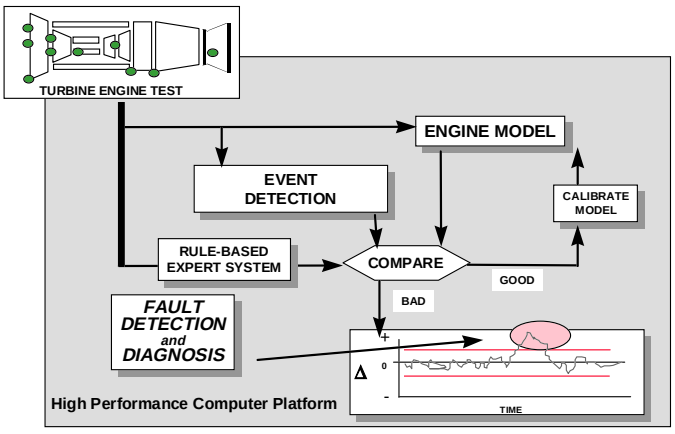

Fig. 1. Online test diagnostic system.

* The research reported herein was performed by the Arnold Engineering Development Center (AEDC), Air Force Materiel 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.

a sound basis, the models must accurately represent the test engine.

Calibration of a model to accurately represent a specific engine is also a labor-intensive task. System identification techniques have commonly been applied to reduce the effort required to calibrate a model and have been used most effectively for calibrating simplified models to measurement sets for which data uncertainties are well de-fined. However, system identification techniques can exhibit numerical divergence as model complexity increases or as data uncertainties increase. Consequently, system identification techniques have not been effectively applied to complex transient models in a developmental test environment. Therefore, to take advantage of the relationships inherent in a component-matching model, a more effective calibration technique is required.

Fault identification capabilities have been demonstrated on systems for which major attributes of the system (e.g., components, configuration, control) remain unchanged and the same operating condition is repeated many times. A variety of approaches provides these capabilities, including expert systems, neural networks, and system identification techniques such as Kalman filtering. The system identification techniques rely on models that are well-defined for model-to-data closure and fault identification while neural network techniques rely on a statistically significant number of samples at a selected operating condition. A component-level model for a developmental engine is not defined sufficiently for these techniques and undergoes frequent change to adapt to changes in the engine's attributes. Additionally, a developmental engine, because of frequent configuration changes, rarely repeats a specific operating condition, reducing the applicability of system identification and neural network approaches. Consequently, a fault identification approach that is able to adapt to engine system changes is required to enable an automated model-based approach to data validation and engine condition monitoring.

The fault identification approach[1] relies on an automated real-time model calibration technique and emulates traditional fault identification processes. The technique focuses on single faults as each occurs rather than on the estimation of an optimal combination of all possible faults. The approach is adaptable to changes encountered in developmental turbine engine testing and relies on a basic component-matching model that represents the engine cycle (e.g., turbofan, turboshaft, turbojet). Industry-accepted engine modeling practices are combined with advanced fault diagnostic algorithms and parallel computer techniques to provide real-time fault identification capability for steady-state and transient engine operation.

OVERVIEW OF MODEL-BASED FAULT IDENTIFICATION APPROACH

To be effective, gas path analysis tools must identify component performance deviations and measurement errors. The model-based fault identification process consists of two main phases. The first phase of the fault identification process relies on a real-time* model-based evaluation of test data to detect a probable fault resulting from measurement errors, engine component events, or a combination of the two. After a fault is detected, automated model simulation studies are performed to diagnose the most probable cause of the fault in near real time. This detailed diagnostic information enables the analysis engineer to assess the relative probability of the fault and, in most cases, quickly identify and verify the actual cause of the fault. If necessary, additional test data may be acquired at previously tested stabilized engine operating conditions to increase the fidelity of the diagnosis. The following sections provide an overview of the adaptive engine model and the model-based fault identification approach.

MODEL DESCRIPTION

A component-level model (CLM), capable of simulating steady-state and transient engine operation, serves as the basis for the fault identification process. The CLM combines the physical relationships that govern engine operation with empirical relationships that describe individual component performance. The result is an adaptable model in which the effects of changes to engine attributes (e.g., components, configuration, controls) are incorporated by making corresponding changes to the model attributes. Additionally, the component-matching approach quantifies the changes to engine performance interrelationships which provide a prediction capability for the fault identification process.

* Real-time speed is defined as the speed required to continuously execute the fault detection algorithms at an average execution rate of 100 Hz.

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- and low-pressure turbines, fan bypass duct, mixer, afterburner, 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 a set of 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.[2] The CLM computes temperature, pressure, mass flow, and rotational speed for each inter-component engine station.

Baseline component performance relationships are an integral element of the CLM; however, they are determined primarily for a baseline component configuration isolated from the engine assembly in a component rig test. Component performance variations that result from operation within the engine assembly are included by applying scaling parameters to the baseline performance relationships. The scaling parameters account for the effects of intercomponent interactions, component modifications, and off-schedule geometry. The scaling parameters are usually applied as scalars to flow, pressure ratio, and efficiency relationships and are defined as a ratio between measured values and baseline values. The scalars for the entire engine, comprised of the scalars from all the individual components, are the primary variables used to calibrate the engine model to a specific measurement set. The method used to determine the values of the scalars for the fault identification process is described in detail in Ref. 1.

FAULT DETECTION

The issue of detecting faults during developmental turbine engine testing with newly developed hardware at never-before-tested conditions is extremely complex. Therefore, the real-time model-based fault identification approach is used to supplement existing approaches that consider on-site calibrations, online monitoring of instrumentation systems, comparison of redundant measurements, steady and non-steady measurements, and measurement to predicted responses.

The fault detection approach relies on interpretation of measured and predicted responses and interrelationships throughout the propulsion system quantified by the CLM. A simultaneous multipoint analysis is used to provide a relative assessment of measurement error and changes in component performance. The multipoint analysis includes the following:

1. Interpretation of differences between predicted and measured aerothermodynamic measurements for the time being considered (to validate modeling assumptions and detect measurement errors);

2. Interpretation of changes in component flows and efficiencies using data immediately prior to and during the time being considered (to detect abrupt faults); and

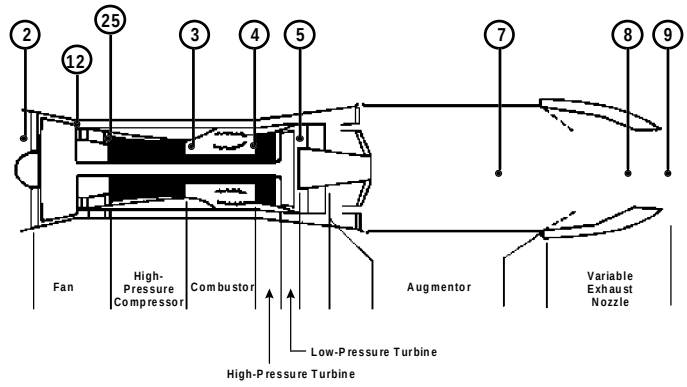

Fig. 2. Station designation for an augmented turbofan engine.

3. Interpretation of changes in component flows and efficiencies using data considerably prior to and during the time being considered (to detect slower faults such as engine degradation or sensor drift).

A weighted root-sum-square fault detection parameter is calculated as a measure of the probability of a fault. The fault probability is a function of time-dependent changes in component flows and efficiencies, the ability to predict test measurements not directly used in the real-time model calibration process, and changes in component flows and efficiencies relative to baseline values. The sensitivity of

the fault detection system is varied to maximize the probability of detecting faults, and to minimize the probability of false alarms. Sensitivity is increased when measurement uncertainties are lower (e.g., stabilized operating conditions at higher power settings). Further increases in sensitivity result at previously tested conditions and configurations (e.g., no hardware or instrumentation system changes). Some reduction in sensitivity is made to account for higher data and model uncertainties during engine starts.

FAULT DIAGNOSIS

Once a fault is detected, an automated model-based methodical search is performed to isolate the most probable cause of the fault. The probability of accurately diagnosing the fault is increased, as previously stated, by emulating the diagnostic process performed by the analysis engineer. Specifically, the automated model-based fault diagnosis approach concentrates on identifying measurement errors which are the most probable faults. If the fault cannot be attributed to measurement error, it is interpreted as a change in component or overall engine performance.

Once a fault is detected, the fault diagnostic system concentrates on identifying individual sensor faults and tries to identify the erroneous measurement and the magnitude of the error. The calibrated CLM (calibrated with data immediately prior to detection of the fault) is used to assess the probability of measurement errors. Each measurement is sequentially perturbed, varying magnitudes to determine the most probable cause of the fault (e.g., fuel flow is perturbed ($\pm$ 5 percent in 0.5-percent increments, then airflow is perturbed, etc.) The model-based diagnostic approach relies on interpretation of measured and predicted responses and interrelationships throughout the propulsion system quantified by the CLM. Due to the highly nonlinear interrelationships throughout the propulsion system, multiple perturbations of varying magnitudes are required to evaluate fault probabilities accurately. The measurement error probability is the inverse of the fault detection parameter. The error probability considers differences between predicted and measured parameters and both rapid and slow changes in component flows and efficiencies.

TARGET PLATFORMS

The data validation process with an embedded engine model requires a significant amount of computing power to execute in a real-time test environment. A network of distributed processors was chosen as the target platform to provide the computing power required for the entire process. A prototype platform was built to demonstrate the effectiveness of a distributed processor approach in providing the required computing power and as an environment for program development. The prototype platform consists of eight 150-MHz Pentium® PC clones interconnected via a dedicated high-speed (100 Mbits/sec) Ethernet network in a cluster configuration. The Pentium® cluster is scaleable since processors may be added or removed as computing requirements vary. As shown in Fig. 3, the high-speed ports can also be connected in a point-to-point scheme (without using a hub) to build an eight-node hypercube.

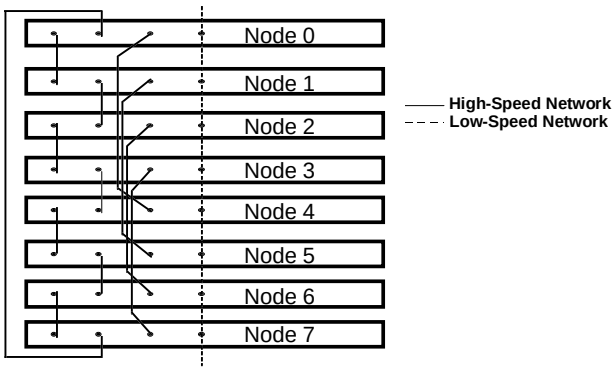

Fig. 3. Eight-processor Pentium® cluster architecture.

Each node of the cluster has four network ports: one of these is a standard 10 base-2 (coaxial) port; the other three are high-speed 100 base-T (twisted pair) ports. The coaxial ports are used for “standard” (remote login, remote shell, network file system, and parallel applications with low communication bandwidth requirements) network traffic while the high-speed ports are used for application-level communications. The high-speed ports are connected in a point-to-point scheme (without using a hub) to build the eight-node cluster.

Each processor runs a copy of either the Linux operating system or Windows NT™ in a multiboot configuration. This allows users to develop and test parallel code under their preferred operating environment. The parallel machine also includes system management tools (small collection of shell scripts added to the standard Linux distribution) to facilitate control of the distributed processors. The Message Passing Interface (MPI)[3] has been

ported to both environments to provide application-level communication services over the high-speed ports. Special user-level network device drivers were developed for MPI to control the high-speed Ethernet ports without the need to use operating system calls. Other parallel programming environments available on the system include PVM and a simple TCP/IP-based communication library.

SOFTWARE ENGINEERING CONSIDERATIONS

All current component-level engine models used at AEDC have been written in FORTRAN. Several reasons for the use of FORTRAN are discussed below:

- Efficiency: Although there are more modern programming languages available, FORTRAN is still one of the best languages for fast numerical computations. The lack of "modern" features (e.g., pointers) makes it relatively easy to write very good optimizing compilers for FORTRAN. Additionally, the emerging High Performance Parallel FORTRAN (HPF) standard promises to simplify creation of parallel FORTRAN programs — at least for certain kinds of parallel architectures and for certain types of problems.

- Historical Reasons: Most of the existing engine model software was developed on mainframes and later ported to personal computers and workstations. On the mainframe the natural development language was FORTRAN.

- Developers: Most engine models are written by aeromechanical engineers who are fluent FORTRAN programmers who want to concentrate on the aeromechanical aspects of the problem instead of software engineering issues.

- Modular Legacy Software: Most models have been developed using existing FORTRAN software libraries and a modular structure applicable to arbitrary engine configurations. The modular structure typically consists of a main program, engine-dependent subroutines and data, component routines, and utility routines. The component routines use generalized logic for determining engine component operating characteristics and for performing aerothermodynamic calculations. Utility routines perform vital tasks such as cycle balancing, data interpolation, and calculation of gas properties. Together, the customized FORTRAN library and the engine-specific data become the model of the given engine.

he programming style used in most existing models relies heavily on global data structures stored in FORTRAN common blocks. Moreover, most software was not intended either for parallel execution or for integration with a graphical user interface. In addition, input and output of the models is typically done with formatted FORTRAN data files. Effective techniques for converting existing FORTRAN software for parallel operation are presented in the following sections.

PARALLELIZATION OF FORTRAN SOFTWARE

The next issue after selecting the hardware platform was to select the methods and evaluate the tools that are available for converting existing software for parallel execution. Generating parallel software from FORTRAN input is a technique that has been used with success for some time. FORTRAN is a primitive language when compared to later languages (e.g., ADA, C, Pascal). This simplicity is, in fact, an advantage when creating parallelizing compilers. In modern languages which support pointers, various subtle interactions are possible between variables — for this reason it is much harder to write a correct and efficient parallel software generator for them.

A number of successful FORTRAN compilers for shared memory Multiple Instruction Stream/ Multiple Data Stream (MIMD) or Single Instruction Stream/Multiple Data Stream (SIMD) architectures have been developed. One of the best examples is the vectorizing FORTRAN compiler for Cray supercomputers which parallelizes on the instruction level. These compilers use techniques such as unrolling loops and assigning the iterations of the loop to different processors. These techniques work well in shared memory architectures, but in distributed systems their applicability is limited. In addition, shared memory parallel computers have limited scalability due to memory contention problems; this effectively limits the possible speedup which can be achieved using these techniques.

Generating good parallel software for scaleable distributed parallel architectures is a more complex task. Simple instruction-level parallelizing does not work very well on these architectures due to communication overhead problems. For these reasons,

parallelizing FORTRAN compilers for distributed architectures are just starting to emerge. These compilers frequently rely on the High Performance FORTRAN (HPF) extensions of the FORTRAN 90 language standard.

Two typical parallel software generators which claim FORTRAN support on distributed architectures are the SAGE[4] toolkit and the FORGE family of compilers.[5] The SAGE toolkit is being developed at the Department of Computer Science, Indiana University, Bloomington, under a DARPA grant. Its primary aim is to implement a parallel version of the C++ language (pC++), but it supports C and FORTRAN as well. The program source is first translated by one of the available compiler front end translators (pC++, C or FORTRAN) into an intermediate form. The output of this phase is a huge dependency file that contains the representation of all of the program's instructions and their interactions. This dependency output in itself is a valuable resource; various browsers can be used to analyze program structure, calling sequences, and data usage. The toolkit also provides software generators that can compile the dependency file for various shared memory and distributed parallel target architectures. The supported distributed target environments include the Intel Paragon system and two widely accepted message passing standards for distributed computing: Parallel Virtual Machine (PVM)[6] and the MPI.[3] The SAGE FORTRAN front-end currently processes only FORTRAN 90 input. Support for HPF extensions is being developed. The current implementation will not parallelize instructions operating on global data.

The FORGE compiler family is a commercial product offered by Advanced Parallel Research, Inc. It contains both interactive program dependency analyzers/browsers and batch mode parallelizing compilers. The interactive tools only support FORTRAN 90, while the batch mode compilers support both FORTRAN 90 and FORTRAN 77. The compilers generate FORTRAN 77 output for either shared memory or distributed architectures. The distributed architecture output contains calls to FORGE's own communication library. This library is implemented as a wrapper on top of commonly available communication interfaces such as PVM, EXPRESS, and TCP-IP. Most of the tools in the compiler family seem to be geared towards the FORTRAN 90 language and HPF, but some FORTRAN 77 parallelization support is provided as well.

In summary, both SAGE and FORGE generate better parallel software from FORTRAN 90 input with HPF directives than from FORTRAN 77 input. However, FORTRAN 77 programs which rely extensively on global variables will probably never be parallelized by automatic tools to the same extent as HPF input. As an alternative to existing parallel software generators, explicit algorithm parallelization and an associated tookit are discussed in the following sections.

EXPLICIT ALGORITHMIC PARALLELIZATION

The other avenue for parallelizing existing FORTRAN software is to understand the algorithm being used and distribute the different functional blocks of the program explicitly to different processors. This also requires that explicit calls to an underlying communication library be made at the points where synchronization and/or data exchange is necessary. Communication libraries such as MPI, PVM, and EXPRESS can be used for this purpose.

The advantage of explicit parallelization is that frequently, a much higher level of parallelization can be obtained than with instruction-level parallelizing compilers. The disadvantage is that explicit parallelization requires changes to the original program and conditional compilation to maintain serial and parallel versions of the software. Also, the algorithms must be analyzed (or reformulated) to find the best decomposition. The dependency analyzer components of some automatic tools can simplify this process.

DEVELOPMENT OF A PARALLELIZATION TOOLKIT

A parallelization toolkit has been developed to aid in explicit software parallelization. Major elements of the toolkit include:

- Performance Analysis: The first task of explicit software parallelization is to identify the software components to parallelize. Various software profiler tools with analyzers for their output can be used for this purpose. Our toolkit contains an analyzer tool which complements the standard UNIX profilers by post-processing their output into a more readable format.

- Dependency Analysis: After the design of the parallel conversion and distribution of the software has been completed, the next task is to

identify the data which must be communicated between the different processors. Two types of dependency analysis tools can be used for this purpose: static and run-time. Development of a static dependency analyzer based on the SAGE toolkit has begun. A significant development effort is required to process arbitrary FORTRAN software. For this reason, run-time dependency analysis is also being considered. This technique is based on a FORTRAN compilation process which uses the AT&T F2C translator. The intermediate C output of the F2C translator is instrumented using preprocessor macros and linked-in debugging libraries. This technique can be used to trace the usage of various common block variables by the different subroutines of the program. A post-processor is then used to generate communication calls based on the global variable access patterns generated by the instrumented program. This technique is not perfect, since it is possible for the instrumented program to avoid some instruction branches and create an incomplete variable access signature. Nevertheless, a significant portion of the parallelized communication software can be developed automatically using this method. It is anticipated that upon completion of the static data dependency tool, the two analysis methods will complement each other.

- Run-Time Libraries: One of our goals for the explicit parallelization work was to develop a method which requires minimal changes to the original software to avoid maintaining separate sequential and parallel versions of the software. This required the development of powerful communication libraries to parallelize the software using a minimal number of additional subroutine calls. We implemented such a communication layer on top of the standard MPI calls. A TCP/IP-based implementation of the communication library is also available. This layer was specifically tuned for parallelization of global variable-oriented FORTRAN software. To accomplish the conversion of the software with minimal changes, a powerful set of communication primitives was developed to work on large data sets grouped by the FORTRAN NAMELIST construct. A dummy version of this library is provided for building sequential versions of the software.

MODEL IMPLEMENTATION

Implementation of the nonlinear model and large amounts of data provided for gas path analysis impose a considerable computational burden, not only in terms of cycle time, but also in terms of divergence control of the nonlinear model. Several approaches were employed to counter these burdens. The first was to simultaneously balance the cycle equations and minimize errors between the model and the data. The second was to optimize the combination of perturbation variables and cycle constraints associated with the nonlinear model. This maximized the accuracy of the simulation and ensured convergence with minimal iteration. Effective use of the partials correction in the modified Newton-Raphson solver eliminated the need to recalculate derivatives and minimized any potential benefits associated with parallelizing the numerical solver.

In conjunction with the above improvements, "task level" parallel processing was used to provide real-time operation. Parallel computing processes shown to greatly enhance the real-time computing capability include simultaneous calibration of component maps, calculation of engine processes (cycle balancing) and integration of dynamic quantities (rotor and gas dynamics) for time-dependent data. Surprisingly, use of model segmentation and distribution algorithms from the "Parallel Processor Engine Model Program"[7]* failed to greatly enhance the real-time capability of the parallel technique and tended to increase the effect of numerical errors and their cumulative effects over time.

SIMULATION INTERFACE

In a related effort, a universal Graphical User Interface (GUI) library has been developed for use with turbine engine model software. The purpose of this GUI is to support the interactive execution of various simulations and provide data visualization capabilities.

The main design goals of the GUI were universal applicability and user extensibility. The same user interface should support several different

* The parallel processing engine model program is a generalized engineering tool intended to aid in the design of parallel processing real-time simulations of turbofan engines. 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.

engine model types and their applications. This was achieved by providing a set of core functions for database services and basic plots and an application program interface (API) for implementation of customized functions. Most of the custom functions are implemented using user-defined FORTRAN software (e.g., engine models, user-defined plot formats, and custom data file interfaces) which can be registered into the basic GUI framework. The main software components of the GUI design are shown in Fig. 4.

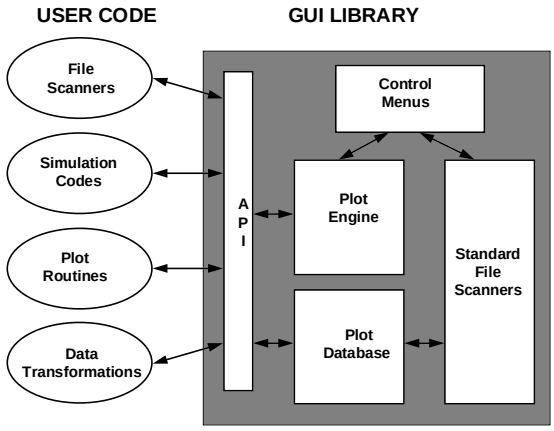

Fig. 4. Graphical User Interface (GUI) architecture.

- Plot Database: This database contains records of measured and simulated variables generated by reading in online data or by running the linked-in simulation software. The database can be searched by parameter names and classes. Each parameter can have several data records associated with it which are keyed by data class names. The Plot Database also allows user-registered simulation subroutines to enter data during a simulation run. In this mode, plots are updated online as the simulation progresses.

- Plot Engine: This component is responsible for generating the plots that are displayed in the plot windows of the user interface. A slightly modified version of the public domain GNUPLOT plotting program was used for this purpose. The Plot Engine operates in either of two modes; it can create standard plots from the data in the Plot Database or display data generated by a user-registered plot subroutine.

- Standard Data File Scanners: The interface provides the capability to scan a few "generic" data file formats. These include binary data files and tabulated data. For more customized file formats, it is possible to register routines using the Scanner Library.

- User Interface Menus and Controls: These components contain the menu options through which the various screens, panels, and menus of the interface are accessed. These control components coordinate the various operations of the other interface services such as selecting the plot window layout, assigning parameters from the Plot Database to the various plot windows for display, and configuring the plot windows. Most of the control services can be invoked via top-level pull-down menus.

- Application Program Interface (API): This layer contains the entry points through which user software can be registered into the user interface. The API is provided for C or C++ programs and for FORTRAN programs compiled with an F2C translator. FORTRAN entry points are implemented as a wrapper around the C entry points to adapt the interface to the FORTRAN environment.

The universal interface library has been implemented for a variety of platforms including IBM PCs running DOS (full screen) or 32-bit Windows applications (WIN32S, Windows™ 95, and Windows NT) and several UNIX workstations with the X window environment (Linux, SGI, and HP).

The universal interface was written in C++ using common interface elements (e.g., buttons and scrollbars) which were adapted from public domain toolkits. Some new widgets were developed because none of the commercially available "portable" toolkits cover all of the targeted platforms (DOS full-screen graphics, Windows, and the X Window system) -- at least not without unacceptable restrictions (like limiting the compiler choices under DOS and Windows or requiring a Motif license for workstations).

DEMONSTRATED INCREASE IN EXECUTION SPEED

The model-based fault detection algorithms executed at about 20 samples/sec on a single 150-MHz Pentium® platform (100 samples/sec required for real-time operation). Attempts to run the software on available high-performance single-proces-

sor workstations did not result in real-time operation. The software was parallelized using a replicated worker approach. Different time samples are sent to different processors for cycle balancing and calculation of engine processes. The scheme uses one central processor to collect engineering unit data from the network, distribute the data between the processors, collect and order the results, identify component performance changes and measurement errors from the reduced data, and run the GUI. Special care was taken to correlate time-dependent processes.

The effectiveness of the Pentium platform and parallelized software was evaluated using an 89-sec test maneuver. Figure 5 shows the execution time as a function of the number of worker processors employed. The results convincingly show that the eight-processor Pentium® system is fully capable of executing the parallelized model-based data validation and fault detection algorithms in real time.

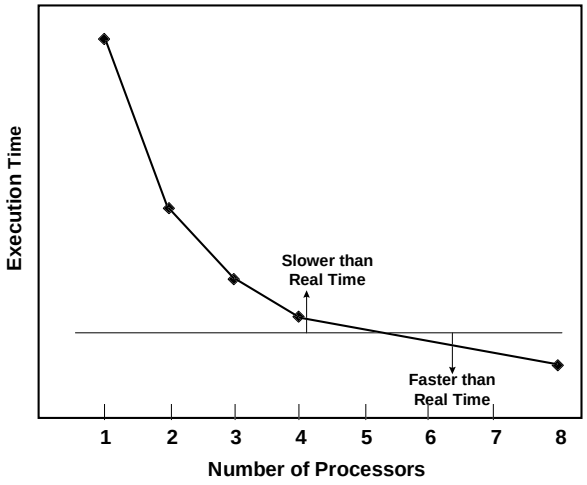

Fig. 5. Pentium® platform effectiveness.

TEST RESULTS

The fault detection and diagnostic algorithms were evaluated using data from two distinctly different modern military turbofan engines undergoing ground test development at simulated altitude test conditions. Data from hundreds of sensors were recorded at rates up to 100 samples/sec to characterize aerothermodynamic engine performance. Both engines were tested over a wide range of steady-state and transient engine operating conditions enabling a comprehensive assessment of the suitability of the approach for developmental engine testing.

The three-step model calibration process[1] is an integral part of the fault identification process. Accurate predictions from the CLM are a key element of the fault detection and diagnostic processes. Figure 6 shows a prediction of engine thrust compared to engine test data after model calibration. Predicted thrust provides an independent measure of the real-time model calibration capability, since measured thrust is not used during the calibration process. The close agreement between measured thrust and predicted thrust demonstrates the accuracy provided by the model calibration capability.

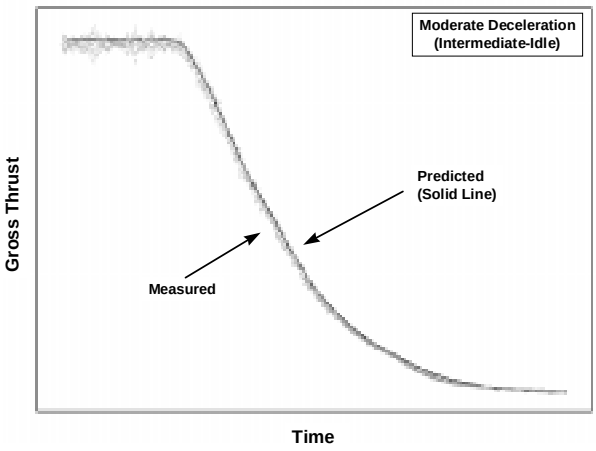

Fig. 6. Predicted and measured gross thrust.

Propulsion control schedules and engine hardware often vary significantly from basic design conditions during developmental engine testing. The situation is compounded by the use of specific propulsion control laws and schedules developed during engine tests to tune overall system performance to match individual mission roles. The capability to identify measurement errors and anomalous changes in component performance during control logic development is a formidable task because changes to the control laws may result in atypical, but valid, engine operation. False alarms and incorrect diagnoses will result unless the control law changes are reflected in the fault identification analyses. For example, potentially anomalous engine behavior or anomalous measurement of engine airflow is indicated in Fig. 7 by an erratic, rather than uniform, increase in engine airflow for a moderate engine acceleration. However, the erratic change in airflow is a result of control law manipulation and represents valid engine operation. The real-time portion of the calibration process, discussed previously, inherently accounts

for the control law changes and results in a confirmation of valid engine operation for the data shown in Fig. 7. The fault identification process correctly reported no faults, thus demonstrating its suitability for developmental engine testing.

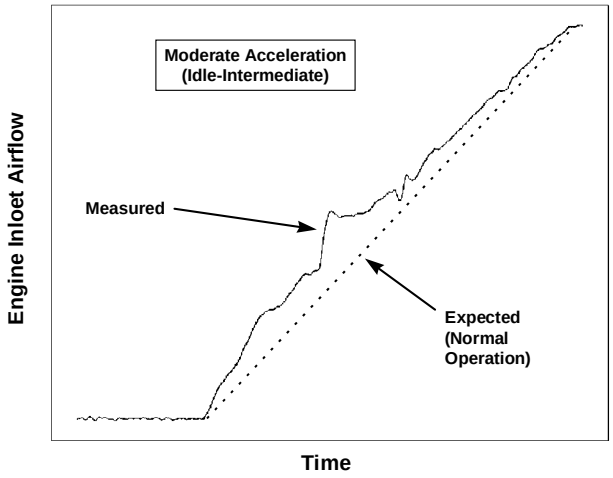

Fig. 7. Effects of propulsion controls on airflow measurements.

Sensor faults identified during transient engine operation using the model-based approach include the identification of a slowly responding pressure measurement used in the determination of engine inlet airflow. The fault was initially identified as an engine airflow measurement anomaly indicated by differences between normal and faulty airflow indications as shown in Fig. 8. The differences are evident near the end of the deceleration and, to a lesser extent, in the steady portion of the data immediately prior to and following the deceleration. Further investigation revealed the root cause as a crimped pressure line which affected the transient airflow measurement. The diagnosis demonstrated the applicability to transient engine operation by identifying a faulty time response that is not detectable during steady-state engine operation.

Data anomalies are equally likely to occur during steady-state and transient engine operation, and the wide range of engine operating conditions enabled detection and diagnosis of a number of sensor faults while demonstrating a low occurrence of false alarms. The fault identification approach is most sensitive during steady-state engine operation at high power settings and is capable of detecting very small (less than 1 percent) errors. Common data anomalies that have been identified using the model-based approach include identification of sensor biases, drifts, level shifts, and excessive noise.

An important function of the fault identification capability is its ability to distinguish between measurement anomalies and engine anomalies. Data for a steady-state engine operating condition were chosen for clarity to illustrate the capability to distinguish between measurement and engine anomalies. For steady-state engine operation, data in Fig. 9 seem to indicate the occurrence of an engine anomaly implied by an abrupt reduction in compressor efficiency and a corresponding reduction in flow capacity. The effects of the observed changes in compressor performance were also seen in other parameters throughout the engine, as would be expected with a deviation in compressor performance. Superficial inspection of the data leads to a conclusion that a compressor anomaly exists.

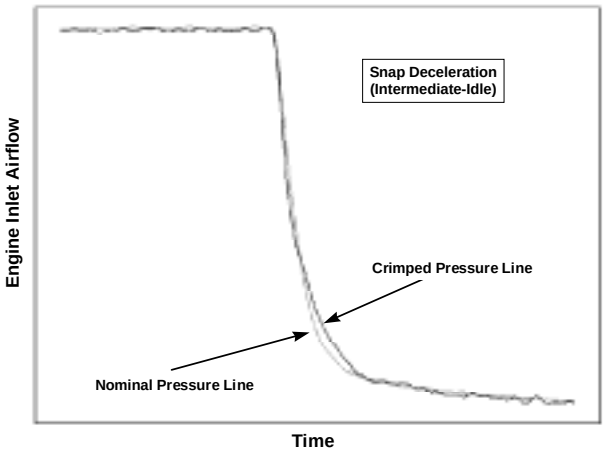

Fig. 8. Effect of slowly responding pressure measurement on aiflow measurement.

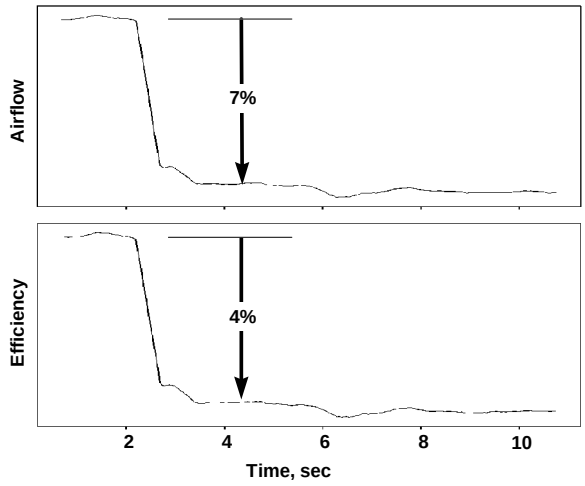

Fig. 9. Abrupt change in compressor performance during steady operation.

However, the fault identification process diagnoses the effects of a faulty P_3 measurement on related cycle parameters, correctly identifies the P_3 measurement anomaly, and distinguishes it from an apparent engine anomaly. The high probability of a P_3 anomaly, as determined by the fault identification process, is shown relative to the probability of other measurements in Fig. 10. In contrast, if the probability of a P_3 error had also been low, a compressor anomaly would have been diagnosed. Independent analysis using traditional methods confirmed the anomalous P_3 diagnosis.

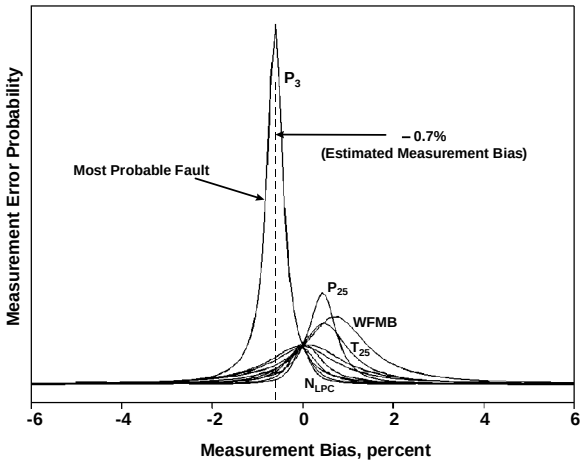

Fig. 10. Fault identification estimate of bias.

Important to note in the anomalous P_3 diagnosis is that the fault was detected early in the event (Fig. 11), clearly exceeding the fault detection threshold. Although the P_3 measurement eventually shifted to 95 percent of its true value (i.e., 5-percent error), the fault was identified while the error was approximately 0.7 percent. The P_3 fault, which was indicative of other faults that were identified, demonstrated the capability of the model-based approach to successfully detect and diagnose sensor anomalies as they occur and distinguish these anomalies from variations in component and overall engine performance.

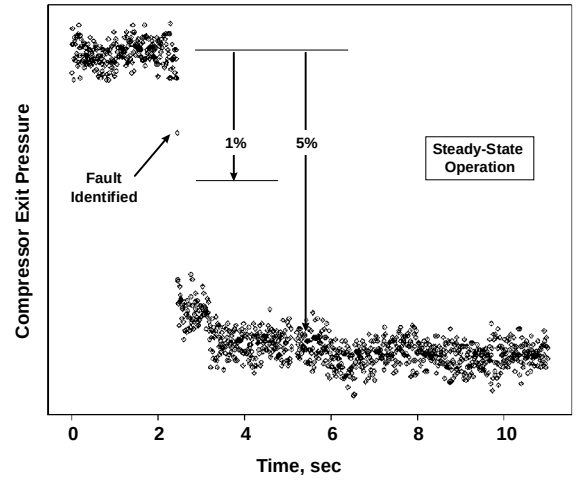

Fig. 11. Abrupt change in compressor exit pressure during steady engine operation.

The technique has been demonstrated during developmental turbine engine testing at simulated altitude conditions to assess the viability of the approach both in terms of the functionality of the model-based approach and the required computational speed. The results indicate that the technique is capable of detecting and diagnosing abrupt changes in measurements.

SUMMARY

The primary goal of the research described in this document was to develop the computational technologies necessary to permit real-time operation of turbine engine modeling technologies in an online test environment. Work has progressed successfully on multiple fronts to permit real-time operation of the model-based fault detection algorithms and GUI on a distributed parallel network. Explicit parallelization of the model-based fault detection and diagnostic algorithms on an eight-processor Pentium platform is shown to provide a higher level of parallelization than with instruction-level parallelizing compilers. The ease of implementation of the parallelization approach and highly satisfactory nature of the numerical results indicate the effectiveness of the distributed network for real-time data processing.

REFERENCES

1. Malloy, D. J., Chappell, M. A., and Biegl, C., "Real-Time Fault Identification for Developmental Turbine Engine Testing," ASME Report 97-GT-141, June, 1997.

2. Chappell, M. A., and McLaughlin, P. W., “An Approach to Modeling Continuous Turbine Engine Operation fro Startup to Shutdown,” *Journal of Propulsion and Power*, Vol. 9, No. 3, May-June 1993.

3. MPI Standard, version 1.1, MPI-2 Forum, June 12, 1994.

4. "Distributed pC++: Basic Ideas for an Object Parallel Language," *Scientific Computing*, Vol. 2, No. 3, Fall 1993.

5. "Automatic FORTRAN Parallelization for Distributed Memory Systems with FORGE Magic/DM," Applied Parallel Research, Inc., Placerville CA, 1992.

6. *PVM: Parallel Virtual Machine -- A Users' Guide and Tutorial for Networked Parallel Computing*, MIT Press, Cambridge, 1994.

7. "Parallel Processor Engine Model Program," NASA-CR-174641, January 1984.