NASA-CR-205332

# PARALLELIZATION OF ROCKET ENGINE SIMULATOR SOFTWARE

# (P.R.E.S.S.)

# RESEARCH SUMMARY REPORT

**Principal Investigator**

**Dr. Ruknet Cezzar, Associate Professor**
**Department of Computer Science**
**Hampton University**
**Hampton, Virginia 23668**
**email: cezzar@cs.hamptonu.edu**

**Technical Officer**

**Dr. Don Noga**
**Mail Stop 501-2**
**NASA Lewis Research Center**
**21000 Brook Park Road**
**Cleveland, Ohio 44135**

**Grant Number: NAG3-1792**

**Grant Period: 3 Years**

**Start date: October 19, 1995**
**End Date: October 18, 1998**

**( September 2, 1997 )**

# TABLE OF CONTENTS

# 1. Background

Parallelization of Rocket Engine System Software (PRESS) project is part of a collaborative effort with Southern University at Baton Rouge (SUBR), University of West Florida (UWF), and Jackson State University (JSU).

The second-year funding, which supports two graduate students enrolled in our new Master's program in Computer Science at Hampton University and the principal investigator, have been obtained for the period from October 19, 1996 through October 18, 1997. The interim progress report dated April 28, 1997 outlines the plans and progress in its relevant sections:

Background
Research Progress Overview
Current Research Activities (Oct 1996 through April 1996)
Difficulties Encountered
New Directions for Second Year Funding
Expected Challenges
Conclusion

The key part of the interim report was new directions for the second year funding. This came about from discussions during Rocket Engine Numeric Simulator (RENS) project meeting in Pensacola on January 17-18, 1997. At that time, a software agreement between Hampton University and NASA Lewis Research Center had already been concluded. That agreement concerns off-NASA-site experimentation with PUMPDES/TURBDES software. Before this agreement, during the first year of the project, another large-scale FORTRAN-based software, Two-Dimensional Kinetics (TDK), was being used for translation to an object-oriented language and parallelization experiments. However, that package proved to be too complex and lacking sufficient documentation for effective translation effort to the object-oriented C++ source code. The focus, this time with better documented and more manageable PUMPDES/TURBDES package, was still on translation to C++ with design improvements.

At the RENS Meeting, however, the new impetus for the RENS projects in general, and PRESS in particular, has shifted in two important ways. One was closer alignment with the work on Numerical Propulsion System Simulator (NPSS) through cooperation and collaboration with LeRC ACLU organization. The other was to see whether and how NASA's various rocket design software can be run over local and intra nets without any radical efforts for redesign and translation into object-oriented source code. There were also suggestions that the Fortran based code be encapsulated in C++ code thereby facilitating reuse without undue development effort. The details are covered in the aforementioned section of the interim report filed on April 28, 1997.

## 2. Research Progress Overview

In this section, we give an overview of progress since the interim report filed in April 1997. Toward this end, the stipend support for one graduate student, Ms. Chenhong Lu has continued for the Summer 1997 for the period from June 1 through July 31, 1996. Although, Ms. Lu's contract was until August 31, 1997, she was transferred to another departmental project one month short of termination of her contract. This transfer was at her request and for better utilization of her talents as a graduate assistant in another project within our department, and did not significantly affect our research efforts on this, P.R.E.S.S., project. As shown in her task assignment for Summer 1997 in Appendix-A, her efforts continued on debugging and understanding the workings TURBDES and PUMPDES software packages, and piecemeal translation into C++ code. To this end, she has translated approximately 2000 lines of source code from VAX FORTRAN to C++ under GNU's g++ compiler on UNIX platform.

Meanwhile, the principal investigator focused primarily on establishing the infrastructure toward the main goal of being able to run NASA's rocket engine software in a distributed manner over local and intra nets. Primarily with cooperation from the NPSS organization, we have made impressive progress on this front as detailed in the next section. Although we feel that the foundation in terms of basic networking tools and packages for distributed use of rocket engine system software has been laid out, the next crucial step of actually demonstrating the viability of such approaches presents some formidable challenges. These topics are detailed and discussed in the next two sections, respectively, and the next crucial step awaits third-year funding of this project.

In summary, while, Ms. Chenhong Lu, graduate research assistant for this project which we refer to as the PRESS project, continued to concentrate her efforts on the previously obtained TURBDES/PUMPDES rocket engine design packages (9), the PI has worked on establishing a foundation for future work toward distributed access and use of NASA's rocket engine design software packages. The PI's work in this new direction can be categorized as:

1. Parallel and distributed computing tools and packages for sharing of rocket engine software across local area and corporate intranets

2. Investigation of C++ wrappers around Fortran code and related issues of mixed-language programming interfaces involving Fortran, C, and C++

3. Checkout and correct installation of Rocket Engine Transient Simulator (ROCETS) on Unix platform to enhance understanding of rocket components and design issues

Further details on these activities and accomplishments are discussed in the next section.

# 3. Current Research Activities

Much of the progress since the interim report of April 1997 has been during the Summer 1997 months. In accordance with the University's course load requirements, with 25% release time, the PI is expected to teach three courses per semester. The normal course load is four courses with up to three distinct preparations. In either case, 180 contact hours with approximately 60 students per semester a required. This helps explain why most of the research progress is during Summer months at this, as well as at other undergraduate institutions whose expressed emphasis on teaching. The following is a discussion of activities since April 1997.

## 3.1 TURBDES/PUMPDES Checkout and Translation to C++

Ms. Lu's efforts involved reconstruction of the missing test input data for various types of rocket engine pump and turbine designs and obtaining meaningful report outputs on performance of viable designs. She has also worked on GASP submodule which provides data on different types of gas propellants used in the design. The GASP submodule, which was missing from the original package, is an "ad hoc" addition to the TURBDES and PUMPDES packages. There are problems with this interface as is discussed in the next section on difficulties encountered.

She was encouraged to use Fortran-to-C translation tool, f2c, in order to first translate the code to C and then redesign and rewrite it in C++. However, before she could do so, it was clear that the code, originally developed in VAX Fortran, was unstructured since it was developed before the advent of structured programming principles. Thus, TURBDES and PUMPDES source code with which she dealt had enormously complicated programming style which predated the structured methods and concepts. For instance, in both TURBDES and PUMPDES source code, there are many many GOTO statements. These numerous GOTOs are forwards as well as backwards making the source code difficult to read and understand. Furthermore, there are unconventional use of shared variables (e.g., FORTRAN COMMONs) for passing parameters to subprograms adding to complexity. Finally, since the code has been developed piecemeal over a period of time, there are many ad hoc patches and revisions.

Thus, before attempting to utilize the tools like f2c, her preliminary attempts involved putting the code in more structured form by replacing the GOTOs with modern loop constructs. Then, she was able to provide the C++ equivalent code in more structured format. In this way, she was able to translate approximately 2000 lines of TURBDES source code before leaving for another project in our department.

## 3.2 Parallel and Distributed Computing Packages

During a teleconference in February 1997, where the PI, Rick Blech, Joe Hemminger, and Angela Quealy were participants, the research in the new direction toward finding ways for distributed access to the fairly rich repertoire of mostly Fortran based rocket design software associated with RENS and NPSS projects. First, it was agreed that the PI be given access to NPSS cluster of computers through an account. Moreover, we discussed tools for distributed

parallel execution of code on different network nodes over local or intra nets. In this regard, as a starting point, we discussed exploring the well known and publicly available standard packages Message Passing Interface (MPI) and Parallel Virtual Machine (PVM). Specifically, we would install and test such tools over Hampton University's Sun Sparcstation network, as has been done for the NPSS LACE cluster of workstations. To this end, Angela Quealy was designated as our contact person at Lewis Research Center. Since then, we have requested and obtained an account on the LACE cluster and, as detailed shortly, were able to successfully install and test these distributed computing tools and their variants. In this regard, Angela Quealy's comments and suggestions, as well as the test demo packages, have been most helpful.

*MPI (Message Passing Interface)*

MPI is a specification for a standard library for message passing. It was defined by the MPI forum, a broadly based group of parallel computer vendors, library writers, and application specialist. As its name implies, it uses message passing protocols as the main paradigm of parallel computations over heterogenous processing nodes (as opposed to some other parallel processing models such as shared memory, channels, generative, etc.). As such, the model is suitable for both local area networks and parallel Multiple Instruction - Multiple Data (MIMD) machines such as the 64-node NCUBE also available, attached to the SGI node at Hampton University's local net. The software which is basically a shared library as mentioned is available from Argonne National Laboratories. Specifically, we downloaded Ver 1.1 which is a minor improvement over Version 1.0 of what is referred to as MPICH for sun4 sparcstation hardware at Hampton University. The CH suffix refers to ch_p4 device describing the processing nodes of the sun4 based local area network. The precursor to MPI was a package named P4, also available from Argonne National Laboratory. The PI had familiarity and experience with this earlier package in conjunction with the development of a parallel processing course at Hampton University through a cooperative arrangement with Argonne back in 1992.

First we downloaded the full distribution mpich.tar.Z, from ftp.mcs.anl.gov, then configured it for ch_p4 device, and finally installed for sun4 and client nodes. Next, we tested the package using mpi-demo package sent from ACLU. We successfully ran the demo programs hello, ring, and laplace. The later, laplace, is available only in Fortran, others are available in Fortran and C. By an email, we happily reported the duplication of results obtained over LACE cluster at our site over sun4 client host machines. Figure 1 shows a typical process group file, pgroup, for the execution of Fortran program laplace over four sparcstations using MPI.

```
apple 0 /users/cezzar/mpi-demos/laplace/fortran/laplace
peach 1 /users/cezzar/mpi-demos/laplace/fortran/laplace
basil 1 /users/cezzar/mpi-demos/laplace/fortran/laplace
orange 1 /users/cezzar/mpi-demos/laplace/fortran/laplace
```

**Figure 1**: Parallel Execution of 'laplace' program on Sun Network Using MPI

Later, we also tested the demo packages using our new account for the LACE cluster at ACLU organization. To augment and complement the available on-line documentation, we also purchased a well known reference book on this package [11].

*PVM (Parallel Virtual Machine)*

PVM is a software tool that enables a collection of heterogenous computers to be used as a coherent and flexible computational resource. The individual computers may be scalar workstations interconnected by a variety of networks, such as ethernet which is our case, fiber distributed data interface (FDDI), and so on. The nodes themselves may be vector computers such as Cray, distributed-memory, or shared-memory multiprocessors. PVM support software executes on each node in a user-configurable pool, and presents a unified computational environment for concurrent applications. In a nutshell, the aim is to use a collection of Unix computers hooked together by some kind of network as a single large parallel computer. The software, which is available free from netlib, has been compiled for many different hardware platforms from Cray to IBM PCs. The PVM library routines allow access from high level languages C, C++, and Fortran. PVM uses the message passing paradigm, as does MPI, with special synchronization mechanisms through barriers and rendezvous). There is transparent handling of massage routing and data conversions heterogenous network environment.

We downloaded the current release, PVM3.3.11, from http://epm.ornl.gov, as well as other variants, xpvm, javapvm, and pvm for Win32. The PVM3.3 package is also available at ACLU's LACE cluster for RS6000 nodes. Then, we successfully installed this package for SUN4 host nodes. Later, we ran the test demo package from ACLU and duplicated the results obtained on the LACE cluster. Figure 2 shows a typical process host file, hostfile, for the execution of Fortran program LAPLACE on four sparcstations using PVM.

```
apple.cs.hamptonu.edu  dx=/users/cezzar/pvm3/lib/SUN4/pvmd3
ep=/tmp_mnt/users/cezzar/pvm-demos/laplace/fortran
wd=/tmp_mnt/users/cezzar/pvm-demos/laplace/fortran

peach.cs.hamptonu.edu  dx=/users/cezzar/pvm3/lib/SUN4/pvmd3
ep=/tmp_mnt/users/cezzar/pvm-demos/laplace/fortran
wd=/tmp_mnt/users/cezzar/pvm-demos/laplace/fortran

orange.cs.hamptonu.edu dx=/users/cezzar/pvm3/lib/SUN4/pvmd3
ep=/tmp_mnt/users/cezzar/pvm-demos/laplace/fortran
wd=/tmp_mnt/users/cezzar/pvm-demos/laplace/fortran

basil.cs.hamptonu.edu dx=/users/cezzar/pvm3/lib/SUN4/pvmd3
ep=/tmp_mnt/users/cezzar/pvm-demos/laplace/fortran
wd=/tmp_mnt/users/cezzar/pvm-demos/laplace/fortran
```

**Figure 2**: Parallel Execution of 'laplace' program on Sun Network Using PVM

***XPVM*** is a GUI which uses the Unix X-Windows facilities. It implements a console and monitor for PVM. Normally, PVM console, in our case PVM3 console, accepts a variety of commands for configuring the system and setting options. It simply gives a prompt as PVM> and then accepts command lines. For instance, a frequently used command is ADDHOST which adds new nodes of a network into the configuration and causes the associated deamon process to start running on the indicated node. Another console command is HALT which halts the system and cleans up by killing the deamons at various network nodes. With XPVM, this is done via a pull-down menu graphically in a pleasing way and the configuration icons are displayed on a window. This pleasing X-Windows based GUI, however, comes at a greater cost in terms of resource usage and complexity. For instance, for XPVM 3.3.0 or later, we had procure and install other packages TCL 7.3 or later (specifically TCL 8.0 beta release) and TK 3.6.1 or later (specifically TK 8.0 beta release). At any rate, after many trials involving the correct build of all these three packages using Unix make facility (specifically Makefile.aimk generator files for Makefile scripts), we were able to successfully install XPVM 1.0 on our system as well. This new package simply provides a more pleasing user interface but still requires the running of the PVM3 package in the background. Nevertheless, it is a dazzling display of the main concepts involving the parallel execution of concurrent programs over multiple network nodes.

***PVM for Win32*** This is the PVM version intended for PC networks under Microsoft Windows in 32-bit real mode. It requires rshd remote shell for Windows 95 boxes and rexecd for Window NT boxes. We have downloaded and experimented with this on our WIN95 PC. However, to get any use of this package one needs a networked cluster of WIN95 or Windows NT personal computers. It should further be noted that, since the original design of PVM has exclusively been based on Unix hosts, its extensions into other platforms is unnatural and awkward. However, as will be discussed next, its extension to cover Java based applications on Unix networks is more promising.

***JavaPVM*** is yet another version, currently 1.01, which allows Java applications to use the parallel execution facilities of PVM. Recall that the current version, PVM3, already supports Fortran 77, C and C++ applications. We see great promise in this version for the obvious reason that Java is becoming immensely popular for distributed computing and is expected to take over C/C++ applications. We have also downloaded and attempted to install this on our SunOS sun4 sparcstation network. JavaPVM requires the GNU make facility and also a Unix based Java virtual machine. The current version of Java, jdk1.1 freely available from SunSoft and other vendors, can only be installed on Solaris based Sun workstations. Therefore, since we were not able to have a Java package on our SunOS based system, it was not possible to experiment with JavaPVM. Finally, it should be noted that there are various other versions of PVM, such as WPVM which is yet another version for Microsoft Windows, and JPVM which is a class library implemented in Java for use with Java.

### 3.3 C++ Wrapping and Fortran-C Interfaces

During the RENS Meeting at Pensacola, Florida, encapsulation (wrapping) of Fortran code in C++ was an important topic. This idea is very appealing since it permits reuse of existing Fortran code in an object-oriented framework. According to this, all that is required is clean interfaces to Fortran based objects from C++ wrappers. In this context, an email from Joe Hemminger provided some clues as to how this could be done. However, due to missing code in the example, we were unable to demonstrate the viability of this approach on our Unix platform. The essence of the suggestion is shown in Figure 3 on the next page. From what is shown in the figure, it is clear that the remainder of Fortran function THERM is missing and the base class of the calling C++ code is missing. More important, the specific compiler and linking process, apparently on the PC platform because of the appearance of the terms 'far' is of paramount importance. Due to the proprietary nature of the package, we were unable to obtain further information on this issue.

Later we obtained a workable sample code from Angela which demonstrates Fortran to C interface where C functions are called from within Fortran programs. On Unix platform, using C compiler cc and Fortran compiler f77, we were able to successfully develop and link this sample as shown in Figure 4 following next page. Looking at this figure, since time and date calls from Fortran are not in the standard intrinsic set of library functions, we supplied the values fictitiously through the loop shown. The calls to sub1 and sub2 are normal Fortran-to-Fortran and C-to-C calls respectively. These are included to see if there are any side effects. The real call is from Fortran main program PROG1 (in prog1.f file) to C function prog2 (in prog2.c). Notice, however, that because Fortran appends an underline character as suffix to its subroutines (and C does not), the C function needs to be exercises under a different name of as prog2_().

This is all well except for two problems. In this example, no values are passed to the Fortran main function. When we attempted to do so, starting with C function sub2() returning double (not shown in the figure), we got inconsistent values on Fortran side. Regardless, the other and more important issue is the direction of the interface. It is from Fortran main to a C subroutine. After all, as is well known and documented for PC platforms, many a vendor provides workable interfaces to C from other HLLs (e.g., BASIC, FORTRAN, Pascal). Indeed, a fairly detailed discussion of such issues are discussed in the chapter entitled Mixed-Language Programming in [13]. The aim, however, is to wrap Fortran code in C or C++ in precisely the opposite direction.

```
========== C++ module getEntropy() in file getentro.cpp ==========
// The following C++ code is calling the FORTRAN function therm. The FORTRAN
// compiler adds an underscore to both the beginning and end of a function
// name. The C++ compiler only adds an underscore to the beginning of the
// name. Thus when referencing the FORTRAN name in the C++ code, you have
// to manually add a trailing underscore.

// Also note that the FORTRAN compiler automatically demotes *everything* to
// lowercase. So in case sensitive C++, you have to use all lower case for
// FORTRAN function, subroutine and argument names.

// Each of the arguments to therm is typecast to the type required by the
// FORTRAN function. This insures that the argument types match. For instance
// _far is currently a single precision number. It has to be cast to a double
// because therm is expecting double precision numbers. This would be good
// practice even if _far was a double on the C++ already.  This way if _far
// is later redefined to be something else, then your code here doesn't break.

//This is some of the C++ code that call this FORTRAN function.
//Definition of the getEntropy function in the header file

  virtual double getEntropy();

//implementation of getEntropy in the body file. The therm_(...) are the
calls to the FORTRAN function.

//------------------------------------------------------
//return the delta entropy
//------------------------------------------------------
NCPReal
ThermFlowStation::getEntropy()
{
   NCPReal PrelTotal = therm_( (int)2, (double)_Ttotal, (double)_far );
   _RgasTotal = therm_( (int)6, (double)_far, (double)_far );
   _entropy = _RgasTotal * log( PrelTotal / _Ptotal );
   return _entropy;
}


============ Fortran function THERM(ID, ARG, FAOLD) in therm.f ======

C* ********************************************************************C
      FUNCTION THERM(ID,ARG,FAOLD)
C--------------------------------------------------------------------C
C
C $Id: therm.f,v 4.2 1996/01/28 19:17:02 saklann Exp $
C
C     PURPOSE: calculates gas properties from built-in cubic spline
C              fits for air and stoichiometric products of combustion
C              with fuels having a hydrogen-to carbon mass ratio of .16.
C
C     FUNCTION DEFINITION: returns the value of the gas property
C
C     CALL ARGUMENTS:
C       ID      - identifies type of independent parameter
C       ARG     - value of independent parameter (see USAGE)
C       FAOLD   - fuel-to-air mass ratio, f/a
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)

C  <<<<<  Note ther rest of this code was missing  >>>>>>>
```

**Figure 3**: An Illustration of C++ Wrapper Code Concept

```
=============== Fortran main program file prog1.f ============
      program prog1

      integer ival1(8), ival2(8)
      print *,'Hello World! (from prog1.f)'

C Fill in the time values instead of calling system time from Fortran
          do i=8,1,-1
            ival1(i) = i
            ival2(i) = 2*i
          end do

C Calculate the first clock value
      time1= float((ival1(5)*3600) + (ival1(6)*60) + ival1(7))
       *       +(float(ival1(8))*.001)
      print *,'TIME=',time1

C   Here's a call to normal Fortran subroutine in sub1.f
      call sub1

C   There's is a call to a C function in prog2.c module function prog
      call prog2

C Calculate the second clock value
      time1= float((ival1(5)*3600) + (ival1(6)*60) + ival1(7))
      time2= float((ival2(5)*3600) + (ival2(6)*60) + ival2(7))
       *       +(float(ival2(8))*.001)

      print *,'TIME2= ', time2
      print *,'Elapsed= ', time2-time1

      stop
      end


=============== Fortran subroutine file sub1.f ================
      subroutine sub1
      print *,'In subroutine sub1 (.f)'
      return
      end


============== C function file prog2.c ====================
#include <stdio.h>
void prog2 ()
{
    printf ("Hello World (from prog2.c)!\n");
    sub2 ();  /* just a normal call to C function in sub2.c */
}


void prog2_ ()  /* this is really what the Fortran main accepts */
{
   prog2 ();
}


============== C function file sub2.c ====================
void sub2 ()
{
    printf ("in subroutine 2 (sub2.c)\n");
}
```

**Figure 4:** An Illustration of Fortran to C Mixed-language Interface

### 3.4 Rocket Engine Transient Simulator (ROCETS)

Although primarily intended for Jackson State University's efforts in developing object-oriented concept diagrams of various rocket engine system components and interrelationships, we have obtained a copy of this software(10) on a 3.5-inch diskette. Later on, we obtained further documentation on various types of fuel turbopumps and rocket cycle engine flow schematics. In addition, there was a general and more detailed explanation of ROCETS for simulating the transient behavior of such components. The documentation was forwarded to us from Dane Elliott-Lewis of the University of Michigan in Ann Arbor at the request of Joe Hemminger (14). The said package also included an original documentation by the developer [15].

Despite this additional documentation, the source code modules given to us on a 3.5-inch diskette were not complete and lacked hierarchical organization for a build. The software is written in a highly structured Fortran code, specifically Lahey's PowerFortran, but we could not build an executable on personal computer or on Unix platforms. Finally, after numerous email exchanges with Elliot-Lewis Dane of NASA Lewis Research Center, we were able to build and test the package on our Unix (SunOS) platform. However, the Unix version, aside from not missing various source module, include, and library components, is organized differently in a modular fashion in different directories. In that sense, it is a bit more complex, but for more readable and comprehensible. It should be noted that this is no small package and altogether has approximately 70 source components and many more intermediate and library components. We were able to run the test case with rctsdemo.run as the configuration file successfully as shown in Figure 5.

```
ENTER RUN FILE NAME
   rctsdemo.run

---- GUESS ROUTINE WAS CALLED ----

          *** SMIT03 CONVERGED IN    2 ATTEMPTS ***

   11 TOTAL PASSES,    1 JACOBIAN EVALUATION(S),    0 BROYDEN UPDATES

err pct state x err tol  1.06E+01  0.000  HTSPLT   1.08709E+02  3.1956 0.0010
err pct state x err tol  8.12E+00  0.237  HTSPLT   1.17433E+02  2.7902 0.0012
err pct state x err tol  4.16E-05  0.965  PTVLM2   3.53186E+02  0.0000 0.0010
TIME=  0.1000  DT= 0.10000  TOTAL= 17  JACOB=  1  BROYD=  5
...

<< about 250 lines of messages on percent errors on convergence >>
...
TIME=  5.0000  DT= 0.10000  TOTAL= 15  JACOB=  0  BROYD= 10
       **************************************************
       ****                                           ****
      *****                 -- ROCETS --              *****
     ******                                         ******
     ******      END OF RUN INPUT FILE ENCOUNTERED  ******
      *****        PROGRAM TERMINATING NORMALLY      *****
       ****                                           ****
       **************************************************
```

**Figure 5**: The Demo Test Run of ROCETS Software (**rctsdemo** executable module)

Figure 6 shows the partial content of the report output file for the above demo test run.

```
************************************************
   1 POINT                          1.000000

 MODULE LIN0 OUTPUT

   2 DPLIN0                         46.80453
   3 PTINLT                         500.0000
   4 PTSPLT                         453.1955

 MODULE LIN1 OUTPUT

 << etc..>>
                    *** STATISTICS IN WRITO1 ***

 PLOT FILE CONTAINING  100 PARAMETERS AND      1 SCANS HAS BEEN WRITTEN

                           --- PLOT FILE ---
                             RCTSDEMO.PLT
                             --- TITLE ---
                           DEMO TEST CASE #1
                             --- MENU ---
 MODULE LIN3 OUTPUT

 << etc.. >>
                    *** STATISTICS IN WRITO1 ***

 PLOT FILE CONTAINING  100 PARAMETERS AND     51 SCANS HAS BEEN WRITTEN

                           --- PLOT FILE ---
                             RCTSDEMO.PLT
                             --- TITLE ---
                           DEMO TEST CASE #1

                             --- MENU ---

       1 TIME           2 AREAVLV1      3 AREAVLV2      4 CFLIN0
       5 CFLIN1         6 CFLIN2        7 CFLIN3        8 CFVLV1
         ..
         ..
      97 WOUTVLM1     98 WOUTVLM2     99 WVLV1       100 WVLV2
```

**Figure 6**: Partial Content of Output Summary Report file **rctsdemo.OUT**

### 3.5 Conference and Workshop Attendance

We have submitted the abstract of our work to the Fourth HBCU Conference, April 9-10, 1997, Cleveland, Ohio. At the time, the travel budget limitations, we were unable to attend and paper post our work. Later, with travel budget adjustments and augmentation from other sources in the department, we were able to attend the Object Expo/Java Expo and Conference, June 2-6, 1997, New York City, New York. At that three-day conference, of which our budget allowed only one-day of conference attendance, there was one presentation [8] that was highly

relevant to our ultimate research goals. The following is a brief discussion of this presentation. Anyone wishing to have a copy of slides for this presentation may request it from us via email to cezzar@cs.hamptonu.edu.

The presentation was made by a European participant, Mr. Thomas Jell, of Siemens Nixdorf, ASW SDP. The discussion focused on an application called SEM (Structured Electronic Manual for Technical Documentation). Its aim is to present structured electronic manuals with multimedia browsing on-line with hyperlinking capabilities for its text over the intranets and the internet. This kind of application is possible by providing CGI scripts on a server; however, it was felt that the server becomes a bottleneck and the user cannot use drag-and-drop for hypertext on his screen. Therefore, the main approach was to provide Java applets at client workstations. As was amply demonstrated over the rest of the conference, Java's capabilities for distributed computing via browsers at client workstations is impressive and is surpassed only by some older C++ based applications with more flexibility. Nevertheless, Java applets could not handle certain heterogenous client nodes via sockets. The application, after extending and exhausting Java applet capabilities using RMI (Remote Method Invocation), required the use of CORBA at the next level to provide hyperlinks to the other heterogenous nodes. First, RMI is kind of mechanism similar to the well know RPC (Remote Procedure Call) on Unix host nodes. CORBA (Common Object Request Broker Architecture) defines a standard for the object oriented interaction of objects in networked heterogenous systems. As is elaborated in the next subsection, this aim practically precludes older high level languages like Fortran. Such languages do not allow object oriented mechanisms like instantiation through dynamic mechanisms, replication, polymorphism, inheritance, and the like. Indeed, the very idea of an "object" as is understood in object-oriented parlance, is alien to such older languages like BASIC, COBOL, FORTRAN. In any case, in the remainder of the presentation, the speaker has emphasized the intricacies involved in using CORBA in conjunction with Java applets. Indeed, the complaint was about CORBA IDL mapping not being the same as distributed Java applications and the difficulty of implementing CORBA objects written in Java. This, despite close similarity of Java to C++ in which CORBA's IDL is written. That, of course, says something about implementation of CORBA objects written in Fortran (if we can even begin to envision such). We elaborate this point further in the next subsection.

*CORBA Revisited*

Indeed CORBA is discussed in conjunction with the design goals of NPSS in [4]. There, some other emerging technologies involving programming languages and libraries, database management systems are discussed and the adherence to standards and development of standards are emphasized. CORBA is embraced as a unifying standard for the open and distributed architecture envisioned for NPSS. Nevertheless, some technical issues involving CORBA's IDL (Interface Definition Language) versus the vast majority of Fortran based application software at NASA is not covered in detail. Indeed, this issue was raised during the aforementioned presentation by the PI, and the response of the presenter was somewhat cautions and pessimistic. The speaker pointed out that CORBA's IDL, having been written in C++, presented difficulties even for a very similar object oriented language Java. There was also a mention that the basic

aim of standards like CORBA and application support packages such as Iona ORBIX was to provide distributed environment for *object-oriented* applications where objects and messages passed amongst them are well defined. Fortran, while being ahead of other high level languages in parallel environments on tightly-coupled multiprocessors and vector machines, simply lacks the object oriented-ness to be included in the emerging standards involving distributed execution and data access. Had CORBA been proposed with the vast Fortran application base in mind, its interface definition language would have allowed inclusion of Fortran based objects (if such a thing can exist). In the work cited, there is a hint that this can be remedied by encapsulation (wrapping) of Fortran (and plain C) code in an object oriented language C++ according to well defined standards and conventions. Aside from the difficulties involving C++ wrapper, this may still not be sufficient for use of CORBA standards.

The next question is then what to do about reusing the vast Fortran based code at NASA and at other scientific government laboratories. Based on our experience, albeit still limited, we offer some ideas and suggestions in this regard in the next section.

### 3.6 Some Ideas on Reuse of Fortran Based Code

This is an important issue since there is a large number of Fortran based engineering applications, some well over 100,000 lines [4]. During the first year funding of this project (P.R.E.S.S.), we had some experience with a very large package, Two-Dimensional Kinetics (TDK). This package consisted of some 282 modules which were developed over close to two decades. More recently, we had closer experience with TURBDES and PUMPDES packages which are similar to TDK but not as large (5 modules each). All this software have been developed in VAX Fortran on DEC VAX VMS platform. In the previous section, we touched upon difficulties in experimenting with these packages on Unix platform. By now, it is amply clear that all three packages, TDK, PUMDES, and TURBDES, predate the structured programming methods and styles and are, in a sense, textbook examples of spaghetti code. On the other hand, we recently also had experience with more modern Fortran based software, namely ROCETS. ROCETS is a fairly large package consisting of about 70 modules. It is written in a much more structured style; so much so that the code is almost self documenting. Then, when we see its adaptation to the Unix platform with separate directories and library modules in each, Make files, and the like, we see an almost ideal case of Fortran application code. Nevertheless, some cross-platform and implementation issues arise even in this case. For instance, ROCETS software was written in Lahey PowerFortran which allowed variable length COMMON blocks, suffixed COMMON declarations, and the like. Most other Fortran implementations on Unix and PC platforms do not allow this. Thus in adapting the package to Unix, some effort was spent in that regard. Nevertheless, we may regard ROCETS software and similar packages as the best of the bunch worth saving and reuse. As for the older source code referred to earlier, the rework and rewriting, even if they are re-written in Fortran gain is a worthwhile consideration. Nevertheless, with the advent of object oriented languages like C++ and now Java, it makes sense to expend the redesign, rework, and rewriting effort on one of these. From Figures 4 and 5 which shows the outputs for a successful test run, we can see

that, aside from the various object oriented features, Fortran lacks a GUI interface even in its best form. The only graphical interface to Fortran is the QuickWin facilities for Microsoft family, but even that interface is not a GUI in the strict sense. At least for those Fortran based software with which we became familiar with, the inputs are painstaking (e.g., carefully crafted namelists, input files, etc.) and the outputs are very large and unfriendly tables of numbers looking like COBOL dumps! A rocket scientist or designer deserves better. We doubt that the rocket engineers and scientists, even if they are sympathetic to Fortran code with which they are familiar, would mind friendly GUI interfaces for both inputs and outputs of "any" rocket design software tool. After all, the building architects don't mind accessing CAD packages of glitter and glamour.

We can restate the above thoughts in a nutshell as: There is Fortran code and there is Fortran code. Further, Fortran's lacks object-oriented-ness which is expected and assumed in new standards such as CORBA. This has important implications for unsing the emerging new technologies. The new technologies and standardization efforts assume the so called "objects" as the main paradigm. There are attempts [6] toward making Fortran object oriented but such efforts should be viewed with skepticism. In conclusion, as far the reuse of Fortran based software is concerned, first we restrict the scope only to those related to rocket design (RENS, NPSS, and related), we propose the following:

1. An inventory of all the available Fortran based source code be made with respect to name, origin, size, location, current use, and the like (we are told that such an inventory already exists)

2. A classification be made with respect to: original development hardware platform (e.g., DEC, HP, etc.) and operating system platform (e.g., VMS, Unix, MS-DOS, Windows, X Windows, etc.)

3. A grading be done on various software packages as to whether they are currently working correctly, and if so, whether in frequent use by whom and for what reason. If the software is dormant, just thousands of source lines sitting somewhere and doing nothing, such packages should be de-graded.

4. The above mentioned grading could be extended to the style in which the Fortran code is written. For instance, in our experience, TDK is D-, PUMPDES is C-, TURBDES is C, ROCETS A-, etc..

5. The lower grade Fortran based code should then be redesigned, reworked, and rewritten in the most fashionable modern day object oriented language. Such decisions which involve biting the bullet now rather than paying heavily in later years must be clearly, firmly, and without hesitation. After all, making the rocket engine design scientists and engineers familiar with more modern languages, programming paradigms, and software technologies is an advantage in this regard.

# 4. Difficulties Encountered

## 4.1 Technical

Aside from the lack of structured programming of TURBDES and PUMPDES software, the interface of both TURBDES and PUMPDES to GASP (Gas Parameters) subsystem is problematic. It should be noted that the original GASP subprogram is missing and in its place there is an "add hoc" test subprogram. The call from the main routines involved parameters which were not in the definition of this subprogram. Later it was discovered that the primary mechanism of parameter passing was through COMMON blocks and that block data in the GASP subprogram was used to supply the parameters describing the propellant gas properties. There are various entry points in the GASP subprogram file gasp.f and these are called from anywhere in the main modules of TURBDES and PUMPDES. Moreover, there were some discrepancies involving the parameters passed back and forth through Fortran COMMON blocks. Finally, the READ and WRITE statements appear anywhere and everywhere in PUMPDES and TURBDES main programs as well as in GASP subprogram. In other words, since the development of this software was piecemeal and preceded structured programming principles, there is no notion of single entry - single exit. Nevertheless, after fixing all these bugs, we were able to obtain very large output files reporting performance data. It is almost impossible, however, to test and verify the correctness of such large outputs.

In some checkout cases, there were error messages involving floating point overflows. These errors stemmed from incorrect initialization of values through GASP interface but also from the disparity of internal representations in VAX VMS versus Sun Unix platforms. The double precision floating point representations used by VAX are proprietary and slightly longer than the IEEE floating point standards used on Sun system (and most other systems these days). Indeed, that problem which cause overflows at unexpected places and times is a formidable one unless the whole software is redesigned with careful consideration of values and variables.

## 4.2 Staffing

We have maintained partial support throughout the first two years for only one graduate student assistant even though the budget allowed for two. The reason for this is the diminishing size of our graduate student population and better opportunities elsewhere for our graduate students. The formal and administrative requirements make it very difficult to hire research assistants from other institutions of higher learning in our area. It is easier to get graduate research assistants from other departments of our School of Science, such as Physics, Chemistry, Mathematics. However, the utter lack of programming skills of such graduate students do not serve our the purpose of our research. In other words, if our research efforts did not heavily involve systems and software, and instead focused on theoretical issues, we would have no such difficulty enlisting graduate student assistants from the other departments.

### 4.3 Equipment and Software

Hampton University's local area network is well protected with no telnet access from outside and no faculty access to /usr/local directory which is used for experimentation in some other systems and sites. Nevertheless, it is hard to argue with the security concerns of the system administrators, and the lack of time devoted to research efforts. The administrative efforts involved in teaching, such as the students accounts and their maintenance, the laboratories and the like, naturally leave less time to research related efforts. For that reason, for instance, the packages MPI, PVM, etc. have all been implemented in PI's account and not in a shared area like /usr/local directory (that is the case with ACLU LACE but the institution and the organization is well suited for it).

## 5. Expected Challenges

Our plan is to continue the checkout of TURBDES and PUMPDES packages and after we are sure of correct outputs, to run these software in parallel over our local net in a distributed manner. Each of these consist of four modules using GASP subsystem in common. There are still some problems with TURBDES-GASP and PUMPDES-GASP interfaces having to do with COMMON areas and setting of gas properties as block data. However, we are certain that the parallel execution of this code over Sun sparcstation NFS will be feasible and successful. We expect to the same with the Unix version of ROCETS software with minimal problems. The next step would normally be demonstration of parallel execution of these packages over wider networks using the LACE cluster as the starting point with MPI and PVM. However, as is discussed at some length in the content, the step beyond this, involving CORBA type of standard interfaces is not very clear.

Based on our experience with mixed-language programming [13] and the limited exposure to the idea of C++ wrappers through email exchanges mentioned in Section 3.2, we are highly skeptical about the realization of C++ wrapper approach for reuse of Fortran based code. The reason for this is that, starting with Microsoft family of languages QuickBasic, QuickC, Fortran 5.1, etc., the high level language vendors provided workable and viable interfaces to C from the other languages which also included TurboPascal on PC platform (and some others, even Turbo Prolog!). However, notice that the direction is from the older languages to the more modern and popular languages like C (or more recently C++). There is a reason for this. As is articulated in the respective Chapter of [13], once can have a BASIC or QuickBasic main program call and use C functions but not the other way around. Traditional languages have more complex preparations of the environment before their subroutines are called and an outside C or C++ main program (or higher level subprogram module) has no way of anticipating such preparations. Further, notice that the only success in this area has so far been accomplished on PC platform because of the same vendor providing a number of different HLLs and common linking utilities. The key to success in this area which may generally be regarded as mixed-language programming support is to provide common object file formats and linking facilities for all, or at least a variety of, high level languages. Moreover, such standards must included

personal computer, workstation, and mainframe platforms. We simply do not see any significant effort or event in that direction. Nevertheless, we will continue to pursue the idea of encapsulation of Fortran code in C++ rigorously since it is a very promising one.

Finally, as discussed in Section 4.2, another challenge is to enlist the assistance of competent graduate student research assistants. With third year funding, we will be able to provide full stipend and tuition assistance for up to three graduate students. Upon third year funding, we will post bulletins and sent emails to all faculty concerning this. Although, we are having difficulty finding qualified students in our small pool of graduate students, we are optimistic and hopeful. If that does not happen, we will seek to support undergraduate students under different pay arrangements.

# 6. Conclusion

With successful installation and demo runs of distributed computing tools such as MPI and PVM, we have laid the ground for further experimentation with parallel execution of Fortran based code lent to us under two separate software agreements. We will continue with our efforts involving parallel execution of the code and the idea of making the existing code conform to the object oriented principles as discussed in the previous section. In case we do not make much progress in the area of C++ wrappers, we will attempt outright rework and translation of the TURBDES and PUMPDES code to C++.

As for the policy issue of what is to be done with NASA's large Fortran based software application base, we have provided some ideas and suggestions in Section 3.5. With respect to the last item there, where we propose porting of software written in Fortran in traditional (non-structured) style, we hope this issue gets serious attention and consideration. After all, although it involves effort which may seem unnecessary, there are certain advantages in reworking and translation of existing software from one source language base to another.

It is important not to view such efforts as software redevelopment from scratch. Redevelopment or new development from scratch entails gathering requirements, providing high and low level software architectures based on those requirements, and the whole gamut of software development life cycle. Reworking as we propose bypasses the requirements step thereby saving much time and effort. In effect, the existing old software serves as requirements (provided it is readable and thoroughly understood). Moreover, reworking of source code presents opportunities of catching faults in the design, modeling, and algorithms. For instance, suppose a formula or table for a certain procedure was not used correctly, that will be uncovered during the rework and translation of the source code.

## REFERENCES


[1] Huzel, D. K. and Huang, D. H. *Modern Engineering for Design of Liquid-Propellant Rocket Engines (Second Printing)*, AIAA, Inc., Washington, D.C. 20024, (1992).

[2] Sutton, G. P. *Rocket Propulsion Elements: An Introduction to Engineering of Rockets (6th Edition)*, John Wiley and Sons, Inc., New York, (1992).

[3] Follen, G. Williams, A., Blech, R, and Drei, D.V. (NASA Lewis), Apel, A. (P&W, East Hartford), Byrd, R. (P&W, West Palm Beach), Gardocki, M. (G.E. Aircraft Engines), Crawford, N. (AlliedSignal Engines), Ashleman, R. (Boeing), McNelly, M. (Allison Engine Co.) "Numerical Propolsion System Simulation Architecture Definition," NASA TM 107343, November 1996.

[4] Williams, A., Follen, G., Claus, R., Blech, R, et. al. "Key Technologies for Implementing An NPSS," Technical Memorandum, NASA Lewis Research Center, February 1997, [DRAFT].

[5] Miller, B., Szuch, J. R., Gauigier, R. E., Wood, J.R. "A Perspective on Future Directions in Aerospace Propulsion System Simulation," Lewis Research Center, NASA TM 102038, 1989.

[6] Reese, D. S., and Luke, E., "Object Oriented Fortran for Development of Portable Parallel Programs, Mississippi State University, NASA Grant NAG3-1073 and NSF cooperative agreement ECD-8907070).

[7] Blech, R.A. and Arpasi, D.J., "An Approach to Real-Time Simulation Using Parallel Processing,", NASA TM 81731, 1981.

[8] Jell, Thomas, "Building Multimedia Application Systems Using Component Technology," WEB APPS Solutions Report W4, Object Expo/JAVA Expo '97, New York City, New York, May 2, 1997 (presentation viewgraphs).

[9] Software Use Agreement for TURBDES and PUMPDES between Hampton University (recipient) and NASA Lewis Research Center, October 2, 1996.

[10] Software Use Agreement for ROCETS between Hampton University (recipient) and NASA Lewis Research Center, May 12, 1997.

[11] Snir, M., Otto, S.W., Huss-Lederman, S., Walker, D.W., Dongarra, J. *MPI: The Complete Reference*, The MIT Press, Cambridge, Massachusetts (Second Printing, 1997).

[12] Geist, A., beguelin, A., Dongarra, J., Jiang, W., Manchek, R., Sunderam, V. *PVM: Parallel Virtual Machine, A User's Guide and Tutorial for Networked Parallel Computing*, The MIT Press, Cambridge, Massachusetts (Third Printing, 1996)

[13] Cezzar, R. *A Guide to Programming Languages: An Overview and Comparison*, Artech House Publishers, Inc., Norwood Mass. 02062, c. 1995 (ISBN 0-89006-812-7).

[14] Elliot-Lewis, D. Cover letter addressed to Dr. Cezzar, dated July 28 1997 with accompanying documents on turbopump components and schematics

[15] Pratt & Whitney, Rocket Engine User's Manual, 31 May 1990, United Technologies Pratt & Whitney Government Engine Business, West Palm Beach, Florida, FR-20284

# APPENDIX A: Task Assignment for Ms. Chenhong Lu for Summer 1997

PROJECT: P.R.E.S.S -- NAG 3 1792
Research Assistant
Duration: June 1 thru August 31, 1997
Student Name: Chenhong Lu
Email address: grlu@cs.hamptonu.edu

---

Tasks for Summer 1997
---

Your mission, if you choose to accept it, shall be:

1) Completely translate all codes in TURBDES and GASP software to C++ and DOCUMENT it. Use CC throughout (unless there is a problem which absolutely requires use of g++).

2) Make reports every 10 days, on the average, mentioning progress in terms of modules, and source lines. Discuss problems outstanding at the end of each emailed report.

3) By July 15, should be able to run a demo on TURBDES package with correct results.

4) By August 15, should complete the entire work with TURBDES and demo both on SunOS Unix system.

5) By August 25, should also submit the entire documentation in plain text form.

6) You may consult with Terrence on certain general issues:

t.a.lawrence@larc.nasa.gov
icetee@cs.hamptonu.edu

Good luck.

---