UAH
NAS8-36955

# FINAL REPORT

# FOR

# FNAS COMPUTATIONAL FLUID DYNAMICS

Principal Investigator: John P. Ziebarth

NASA/Marshall Space Flight Center

NAS8-36955/D.O. 17

December 19, 1990

Consortium for Computational Fluid Dynamics

56

Under this NASA contract the University of Alabama in Huntsville (UAH) was to provide the following:

1. identification of critical flow problems related to propulsion systems and identify computational fluid dynamics (CFD) resources for application to these problems;

2. promotion of CFD technology from research centers to technology development centers and direct the application of CFD as a design tool;

3. encourage industry and university participation in CFD/propulsion research activities through their own internal and external funds;

4. provide peer review to CFD/propulsion programs;

5. direct the verification and validation of CFD methodologies as they are applied to propulsion problems.

Items 1-4 of the proposed UAH tasks above have been accomplished through the establishment of four Principle Investigator Working Groups or Technology Teams under the Consortium for Computational Fluid Dynamics Application in Propulsion Technology (see Figures 1 and 2) established by the CFD Branch at NASA/MSFC. Coordination of these activities has been through the Consortium for Computational Fluid Dynamics (CCFD), established at and by UAH for this purpose. The four established technology teams are:

- Turbine Stage Technology Team

- Pump Stage Technology Team

- Combustion-Driven Flow Technology Team

- Complex Flow Paths Technology Team

The technical management of each of these teams is controlled by a staff member from the CFD Branch at NASA/MSFC and the members of each technical team are from the member organizations in the CCFD (see Figure 3).

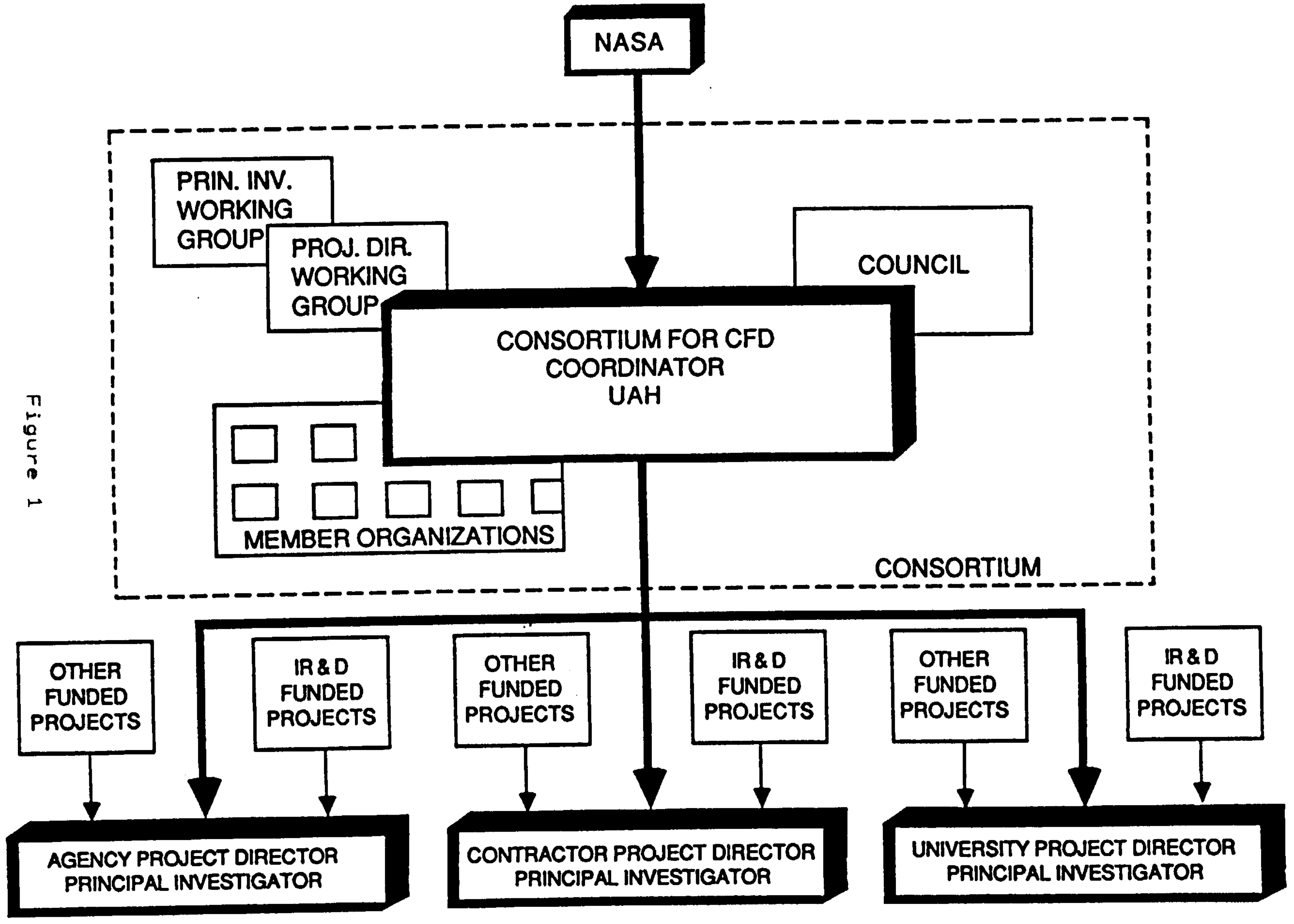
The published objectives and tasks of the CCFD are listed below.

Objectives:

1. Focus CFD applications in propulsion
   a. ETO
      i. Direct baseline program towards improved accuracy, stability and efficiency

## CONSORTIUM FOR CFD APPLICATIONS IN PROPULSION TECHNOLOGY

### ORGANIZATIONAL STRUCTURE



5-799-8-24

Figure 1

## CONSORTIUM FOR CFD APPLICATIONS IN PROPULSION TECHNOLOGY

### ORGANIZATIONAL FUNCTIONS

| | |
|---|---|
| ● CONSORTIUM | — DIRECT, COORDINATE, AND MAINTAIN A CONTINUING FLEXIBLE PLA FOR THE OVERALL ACTIVITIES OF THE CFD PROPULSION COMMUNITY |
| ● COUNCIL | — PROVIDE OVERSIGHT, REVIEW AND RECOMMENDATIONS FOR THE CONSORTIUM ACTIVITIES |
| ● COORDINATOR | — COORDINATE CONSORTIUM ACTIVITIES AND INTERFACE AMONG CONSORTIUM MEMBERS |
| ● PROJECT DIRECTOR WORKING GROUPS | — DIRECT AND MAINTAIN OVERALL CONTINUITY WITHIN THE TOTAL SCOPE OF CONSORTIUM ACTIVITIES |
| ● PRINCIPLE INVESTIGATOR WORKING GROUPS | — DIRECT SPECIFIC TECHNICAL ISSUES WITHIN THE OVERALL SCOPE OF CONSORTIUM ACTIVITIES AND PROVIDE TECHNICAL DIRECTION |
| ● MEMBER ORGANIZATIONS | — PROVIDE KNOWLEDGE, EXPERIENCE AND FACILITATE THE WORK |

5-1006-8-16

Figure 2

## CONSORTIUM FOR CFD APPLICATIONS IN PROPULSION TECHNOLOGY

### INITIAL MEMBERS

| AGENCIES | CONTRACTORS | UNIVERSITIES |
|---|---|---|
| NASA AMES RESEARCH CENTER | CFD RESEARCH CORPORATION | MISSISSIPPI STATE UNIVERSITY |
| NASA LEWIS RESEARCH CENTER | CHAM OF NORTH AMERICA, INC. | UNIVERSITY OF ALABAMA |
| NASA MARSHALL SPACE FLIGHT CENTER | LOCKHEED MISSILE & SPACE COMPANY | UNIVERSITY OF ALABAMA, HUNTSVILLE |
| | SCIENTIFIC RESEARCH ASSOCIATES, INC. | UNIVERSITY OF ILLINOIS |
| | SOFTWARE ENGINEERS, CONSULTANTS, ANALYSTS | UNIVERSITY OF TENNESSEE |
| | UNITED TECHNOLOGY RESEARCH CORPORATION | UNIVERSITY OF TENNESSEE SPACE INSTITUTE |
| | COMPUTATIONAL MECHANICS CO., INC. | UNIVERSITY OF TEXAS |
| | COMPUTATIONAL MECHANICS CORP. | |

**ENGINE CONTRACTORS**

AEROJET TECHSYSTEMS

PRATT & WHITNEY ENGINEERING DIVISION

PRATT & WHITNEY ENGINEERING DIVISION - SOUTH

ROCKWELL INTERNATIONAL CORPORATION - ROCKETDYNE DIVISION

5-800-8-24

Figure 3

   b. CST:
      i. Stimulate CFD validation towards propulsion flows
      ii. Direct applications codes toward design tools and advanced hardware technology concepts

2. Identify national CFD propulsion requirements

3. Stimulate a forum for government, industry, and university interactions

4. Encourage industry to participate in CFD development with IRAD funds

5. Provide synergism in the CFD community

6. Provide peer review of CFD programs

Tasks:

1. Develop a plan to apply CFD to current and future propulsion systems
   a. Identify and rank critical flow problems related to propulsion systems
   b. Identify national CFD related resources
   c. Define high performance computing requirements to accomplish CFD for propulsion applications

2. Direct CFD technology development to propulsion applications

3. Assess and validate CFD applications in propulsion systems
   a. Develop evaluation criteria
   b. Define and implement benchmark validation
   c. Define and implement validation tests

4. Direct the application of CFD design tools towards advanced hardware technology concepts

5. Accelerate the transfer of CFD technology from universities and research centers to industry and hardware development centers

Three of the Technology Teams are well established and have been meeting for about one year; the fourth, the Complex Flow Paths Technology Team, is in start-up phase. These teams have been very useful in identifying current design issues and promoting CFD solutions to these problems. The quarterly meetings have been useful in driving the progress of the work and coordinating the scheduling of results by various organizations.

Items 1 and 5 of the proposed UAH tasks mentioned previously have been carried out by the principal investigator, one research associate and four graduate students at UAH. Three specific prcjects were accomplished to support these tasks.

One effort involved geometric modeling and numerical grid generation using a software tool called GENIE. This work was carried out in collaboration with an engineer at MSFC. The explanation of how to use this software and a series of computational results are given in Appendix 1.

Another effort to support the CFD Branch has been the analysis and optimization of a large CFD code, ROTOR. The results of this investigation and the analysis of ROTOR are included in Appendix 2.

A third effort has been a validation of an equilibrium chemistry model. The results of this analysis are included in Appendix 3.

## Recommendations

It was agreed between UAH and MSFC that during the initial implementation of the four technology teams that the development of the Consortium Council (see Figures 1 and 2) be delayed.

It is recommended that the implementation of the Council be considered for 1991. The Technology Teams (Principal Investigator Working Groups) will be well established by then and the overall propulsion program will have benefited from their efforts. This would also be the appropriate time to establish a communication network among government, industry and university consortium members. This network would include newsletters and brochures.

## APPENDIX 1

## INTRODUCTION

This report explains the activities that were carried out in order to generate a grid on the IRIS Silicon Graphics workstation. The IRIS is a graphics workstation that allows the user to see a graphical representation of a solution for a particular problem. The grid generation tool that was used is called GENIE. GENIE is a 2-D and 3-D geometry modeling and grid generation tool that is available on the IRIS workstation.

## IMPLEMENTATION

To access the IRIS workstation, the user must input the login number and the password. The syntax for this interaction are:

| Prompt | Input |
|---|---|
| login: | Enter login number and return |
| password: | Enter password and return |

The user gets a prompt on the screen that contains the name of the workstation with a number. The syntax of this prompt is:

```
[ (deckard) or (rachel) ].logname#% ]
```

In order to differentiate the different workstations that are available, different names were given to them. The names of the ones that are available are deckard and rachel. For more information on the naming convention, the system administrator should be consulted.

To run GENIE on the workstation, the user command is genie. The IRIS workstation runs on the UNIX operating system. This means that all commands will be written in lower case letters because of UNIX sensitivity to upper case letters.

As an example of how to use this tool, a 51 x 21 90 degree duct is generated. The planes for this grid are:

| Streamwise (l-value) | Planes |
|---|---|
| 1 | -0.5D |
| 3 | -0.25D |
| 5 | 0 deg.D |
| 15 | 30 deg.D |
| 25 | 60 deg.D |
| 30 | 77.5 deg.D |
| 34 | 90 deg.D |
| 36 | 0.25D |
| 37 | 0.4D |
| 51 | 2.5D |

GENIE is a menu-driven program. The user defines the problem domain through interaction with the program. To define the boundaries for a particular geometry, the first item (defining the boundaries) is selected. After this, the second item (creating the grid patches) is chosen in order to create the patches that are needed to generate the grid. In order to plot, the user picks the plotting item in the menu. All the data that will be required by the program in order to run makes up the data file. By default, this is stored in fort.20. In order to prevent this file from being overwritten the next time a user runs GENIE, the file can be renamed or copied to another file using the UNIX command to move (mv file1 fine2) or copy (cp file1 file2). The data file can be edited by using the editors that are available on the workstations. Once the user finishes the interaction wi the plotting routine, a graphical solution to the problem that was defined by the user is viewed on the screen. The output file (solution) is stored by default in fort.47. This is a binary file. An example of a data file can be found in Part A.

To plot the solution for the grid mentioned above, the steps that were followed are located in Part B. The plotting package used was Plot3d_3.5. Plot3d_3.5 is a 2-D and 3-D plotting package from NASA/Ames. The next step that was taken was stacking the 2-D solution into 3-D. A gridstack program was written in order to accomplish this function. Since the output file that was generated by running GENIE was a binary file, this file had to be converted to a formatted file. In order to do this, Plot3d_3.5 was accessed again. The binary file was then read by inputting

```
re/bin/x=fort.47     .
```

The syntax of the command to convert a binary file to a formatted file is

```
list xyz/formatted/x=outputfile     .
```

The output file is specified by the user. This output file will be the input file to the gridstack program. The gridstack program is called gridstack.f. The program is written in FORTRAN. The program serves as a template in the sense that a user can run a different grid simply by changing the dimension size to reflect the size of the grid. The grid is stacked in the Z-direction using spacing from the Y-direction at inlet plane (l=1). This program is located in Part C. This program was used to stack 51 x 31, 51 x 41, 101 x 41, 101 x 61, and 101 x 81 grids. The gridstack program is compiled by using the compile command available on the IRIS. This command is

```
f77 filename     (in this case, gridstack.f)     .
```

To execute the file, the execute command is used. For this particular program, the execute command is

```
gridstack.f -o gridstack     .
```

For more information on these two commands, the system administrator should be consulted. Using the gridstack routine, the input file in this case was called induct.fmt and the output file outduct.fmt.

After compiling and executing the gridstack program, Plot3d_3.5 was accessed. Since the output file created by executing the gridstack program was a multigrid, the read command was

```
re/mgr/for/outduct     .
```

The sequence ⌞ commands that followed are located in Part D. The different plots that are generated can be found in Part E.

## CONCLUSION

GENIE was used to do geometry modeling and grid generation in support of engineering analysis at MSFC. It is recommended that the newer version of GENIE be implemented on the Silicon Graphics 4-D system. This system is not currently available, but its acquisition is highly recommended.

## PART A

```
PROMPTING
vito
NUMBER OF POINTS IN THE I DIRECTION =  51
NUMBER OF POINTS IN THE J DIRECTION =   21
ACTIVITY OPTION  1: DEFINING BOUNDARIES
INDICES ARE    1 51   1   1  FOR BOUNDARY  1
NUMBER OF SEGMENTS=  9
   1   3 ARE THE INDICES OF SEGMENT  1 OF BOUNDARY  1
SEGMENT TYPE   2  STRAIGHT LINE
LINE OPTION  1 USING 2 END POINTS
INPUT POINTS ARE        -.5000     -1.8000  AND        -.2500     -1.8000
PACK OPTION=0  EVEN SPACING
   3   5 ARE THE INDICES OF SEGMENT  2 OF BOUNDARY  1
SEGMENT TYPE   2  STRAIGHT LINE
LINE OPTION  1 USING 2 END POINTS
INPUT POINTS ARE        -.2500     -1.8000  AND         .0000     -1.8000
PACK OPTION=0  EVEN SPACING
   5  15 ARE THE INDICES OF SEGMENT  3 OF BOUNDARY  1
SEGMENT TYPE   4  CIRCLE
CIRCLE OPTION 3 USING 3 POINTS
FIRST END POINT IS        .0000     -1.8000
OTHER END POINT IS        .9000     -1.5588
THIRD POINT ON CIRCLE IS        .4659     -1.7387
PACK OPTION=0  EVEN SPACING
NO, GOING COUNTERCLOCKWISE
  15  25 ARE THE INDICES OF SEGMENT  4 OF BOUNDARY  1
SEGMENT TYPE   4  CIRCLE
CIRCLE OPTION 3 USING 3 POINTS
FIRST END POINT IS        .9000     -1.5588
OTHER END POINT IS       1.5588      -.9000
THIRD POINT ON CIRCLE IS       1.2728     -1.2728
PACK OPTION=0  EVEN SPACING
NO, GOING COUNTERCLOCKWISE
  25  30 ARE THE INDICES OF SEGMENT  5 OF BOUNDARY  1
SEGMENT TYPE   4  CIRCLE
CIRCLE OPTION 3 USING 3 POINTS
FIRST END POINT IS       1.5588      -.9000
OTHER END POINT IS       1.7573      -.3896
THIRD POINT ON CIRCLE IS       1.6751      -.6597
PACK OPTION=0  EVEN SPACING
NO, GOING COUNTERCLOCKWISE
  30  34 ARE THE INDICES OF SEGMENT  6 OF BOUNDARY  1
SEGMENT TYPE   4  CIRCLE
CIRCLE OPTION 3 USING 3 POINTS
FIRST END POINT IS       1.7573      -.3886
OTHER END POINT IS       1.8000       .0000
THIRD POINT ON CIRCLE IS       1.7884      -.2038
PACK OPTION=0  EVEN SPACING
NO, GOING COUNTERCLOCKWISE
  34  36 ARE THE INDICES OF SEGMENT  7 OF BOUNDARY  1
SEGMENT TYPE   2  STRAIGHT LINE
LINE OPTION  1 USING 2 END POINTS
INPUT POINTS ARE       1.8000       .0000  AND       1.8000       .2500
PACK OPTION=0  EVEN SPACING
  36  37 ARE THE INDICES OF SEGMENT  8 OF BOUNDARY  1
```

```
SEGMENT TYPE    2  STRAI  .T LINE
LINE OPTION  1 USING 2 END POINTS
INPUT POINTS ARE         1.8000        .2500   AND       1.8000        .4000
PACK OPTION=0  EVEN SPACING
  37 51 ARE THE INDICES OF SEGMENT  9 OF BOUNDARY  1
;EGMENT TYPE    2  STRAIGHT LINE
LINE OPTION  1 USING 2 END POINTS
INPUT POINTS ARE         1.8000        .4000   AND       1.8000       2.5000
PACK OPTION=0  EVEN SPACING
INDICES ARE     1 51  21  21  FOR BOUNDARY  2
NUMBER OF SEGMENTS=  9
   1   3 ARE THE INDICES OF SEGMENT  1 OF BOUNDARY  2
SEGMENT TYPE    2  STRAIGHT LINE
LINE OPTION  1 USING 2 END POINTS
INPUT POINTS ARE         -.5000     -2.8000   AND       -.2500      -2.8000
PACK OPTION=0  EVEN SPACING
   3   5 ARE THE INDICES OF SEGMENT  2 OF BOUNDARY  2
SEGMENT TYPE    2  STRAIGHT LINE
LINE OPTION  1 USING 2 END POINTS
INPUT POINTS ARE         -.2500     -2.8000   AND        .0000      -2.8000
PACK OPTION=0  EVEN SPACING
   5  15 ARE THE INDICES OF SEGMENT  3 OF BOUNDARY  2
SEGMENT TYPE    4  CIRCLE
CIRCLE OPTION 3 USING 3 POINTS
FIRST END POINT IS        .0000     -2.8000
OTHER END POINT IS       1.4000     -2.4249
THIRD POINT ON CIRCLE IS        .7247      -2.7046
PACK OPTION=0  EVEN SPACING
NO, GOING COUNTERCLOCKWISE
  15  25 ARE THE INDICES OF SEGMENT  4 OF BOUNDARY  2
SEGMENT TYPE    4  CIRCLE
CIRCLE OPTION 3 USING 3 POINTS
FIRST END POINT IS       1.4000     -2.4249
OTHER END POINT IS       2.4249     -1.4000
THIRD POINT ON CIRCLE IS       1.9799      -1.9799
PACK OPTION=0  EVEN SPACING
NO, GOING COUNTERCLOCKWISE
  25  30 ARE THE INDICES OF SEGMENT  5 OF BOUNDARY  2
SEGMENT TYPE    4  CIRCLE
CIRCLE OPTION 3 USING 3 POINTS
FIRST END POINT IS       2.4249     -1.4000
OTHER END POINT IS       2.7336      -.6060
THIRD POINT ON CIRCLE IS       2.6052      -1.0262
PACK OPTION=0  EVEN SPACING
NO, GOING COUNTERCLOCKWISE
  30  34 ARE THE INDICES OF SEGMENT  6 OF BOUNDARY  2
SEGMENT TYPE    4  CIRCLE
CIRCLE OPTION 3 USING 3 POINTS
FIRST END POINT IS       2.7336      -.6060
OTHER END POINT IS       2.8000       .0000
THIRD POINT ON CIRCLE IS       2.7820       -.3170
PACK OPTION=0  EVEN SPACING
NO, GOING COUNTERCLOCKWISE
  34  36 ARE THE INDICES OF SEGMENT  7 OF BOUNDARY  2
SEGMENT TYPE    2  STRAIGHT LINE
```

```
LINE OPTION  1 USING 2 END POINTS
INPUT POINTS ARE       2.8000       .0000  AND       2.8000       .2500
PACK OPTION=0  EVEN SPACING
  36  37 ARE THE INDICES OF SEGMENT  8 OF BOUNDARY  2
SEGMENT TYPE   2  STRAIGHT LINE
LINE OPTION  1 USING 2 END POINTS
INPUT POINTS ARE       2.8000       .2500  AND       2.8000       .4000
PACK OPTION=0  EVEN SPACING
  37 51 ARE THE INDICES OF SEGMENT  9 OF BOUNDARY  2
SEGMENT TYPE   2  STRAIGHT LINE
LINE OPTION  1 USING 2 END POINTS
INPUT POINTS ARE       2.8000       .4000  AND       2.8000      2.5000
PACK OPTION=0  EVEN SPACING
INDICES ARE    1   1   1  51   FOR BOUNDARY   3
NUMBER OF SEGMENTS=  1
SEGMENT TYPE   2  STRAIGHT LINE
LINE OPTION  1 USING 2 END POINTS
INPUT POINTS ARE       -.5000     -1.8000  AND       -.5000     -2.8000
PACK OPTION=7  PACKED ON BOTH ENDS USING HYPERBOLIC TANGENT STRETCHING
WITH SMALLEST INTERVAL=   .0025000
 SMALLEST INTERVAL IN SECOND SECTION =  .0025000
INDICES ARE  51 51   1  21  FOR BOUNDARY  4
NUMBER OF SEGMENTS=  1
SEGMENT TYPE   2  STRAIGHT LINE
LINE OPTION  1 USING 2 END POINTS
INPUT POINTS ARE       1.8000      2.5000  AND       2.8000      2.5000
PACK OPTION=7  PACKED ON BOTH ENDS USING HYPERBOLIC TANGENT STRETCHING
WITH SMALLEST INTERVAL=   .0025000
 SMALLEST INTERVAL IN SECOND SECTION =  .0025000
ACTIVITY OPTION  2: CREATING GRID PATCHES
INDICES ARE    1 51   1  21  FOR PATCH  1
ACTIVITY OPTION  3: CREATING AUTOPATCHES
ACTIVITY OPTION  5: PLOTTING
  1 PLOTS IN PLOT SET 1
OUTER BORDER TO BE DRAWN ON PLOT       1
  1 PATCHES TO BE DRAWN ON PLOT  1
INDICES ARE    1 51   1  21  FOR PATCH  1 ON PLOT  1
ACTIVITY OPTION  8: OUTPUTTING FILE(S)
YES, GRID IS TO BE OUTPUT
 YES,INPUTS ARE TO BE OUTPUT
```

```
PROMPTING
vito
NUMBER OF POINTS IN THE I DIRECTION =  101
NUMBER OF POINTS IN THE J DIRECTION =   81
\CTIVITY OPTION  1: DEFINING BOUNDARIES
.NDICES ARE     1 101    1    1   FOR BOUNDARY  1
NUMBER OF SEGMENTS=  9
   1    5 ARE THE INDICES OF SEGMENT  1 OF BOUNDARY  1
SEGMENT TYPE    2  STRAIGHT LINE
LINE OPTION  1 USING 2 END POINTS
INPUT POINTS ARE         -.5000      -1.8000   AND          -.2500      -1.8000
PACK OPTION=0  EVEN SPACING
   5    9 ARE THE INDICES OF SEGMENT  2 OF BOUNDARY  1
SEGMENT TYPE    2  STRAIGHT LINE
LINE OPTION  1 USING 2 END POINTS
INPUT POINTS ARE         -.2500      -1.8000   AND           .0000      -1.8000
PACK OPTION=0  EVEN SPACING
   9   29 ARE THE INDICES OF SEGMENT  3 OF BOUNDARY  1
SEGMENT TYPE    4  CIRCLE
CIRCLE OPTION 3 USING 3 POINTS
FIRST END POINT IS         .0000      -1.8000
OTHER END POINT IS         .9000      -1.5588
THIRD POINT ON CIRCLE IS          .4659      -1.7387
PACK OPTION=0  EVEN SPACING
NO, GOING COUNTERCLOCKWISE
  29   49 ARE THE INDICES OF SEGMENT  4 OF BOUNDARY  1
SEGMENT TYPE    4  CIRCLE
CIRCLE OPTION 3 USING 3 POINTS
FIRST END POINT IS         .9000      -1.5588
OTHER END POINT IS       1.5588        -.9000
HIRD POINT ON CIRCLE IS         1.2728      -1.2728
PACK OPTION=0  EVEN SPACING
NO, GOING COUNTERCLOCKWISE
  49   59 ARE THE INDICES OF SEGMENT  5 OF BOUNDARY  1
SEGMENT TYPE    4  CIRCLE
CIRCLE OPTION 3 USING 3 POINTS
FIRST END POINT IS       1.5588        -.9000
OTHER END POINT IS       1.7573        -.3896
THIRD POINT ON CIRCLE IS        1.6751       -.6597
PACK OPTION=0  EVEN SPACING
NO, GOING COUNTERCLOCKWISE
  59   67 ARE THE INDICES OF SEGMENT  6 OF BOUNDARY  1
SEGMENT TYPE    4  CIRCLE
CIRCLE OPTION 3 USING 3 POINTS
FIRST END POINT IS       1.7573        -.3886
OTHER END POINT IS       1.8000         .0000
THIRD POINT ON CIRCLE IS        1.7884       -.2038
PACK OPTION=0  EVEN SPACING
NO, GOING COUNTERCLOCKWISE
  67   71 ARE THE INDICES OF SEGMENT  7 OF BOUNDARY  1
SEGMENT TYPE    2  STRAIGHT LINE
LINE OPTION  1 USING 2 END POINTS
INPUT POINTS ARE        1.8000        .0000   AND        1.8000        .2500
PACK OPTION=0  EVEN SPACING
  71   73 ARE THE INDICES OF SEGMENT  8 OF BOUNDARY  1
```

```
SEGMENT TYPE    2  STRAIGHT LINE
LINE OPTION  1 USING 2 END POINTS
INPUT POINTS ARE        1.8000        .2500   AND        1.8000        .4000
PACK OPTION=0  EVEN SPACING
   73 101 ARE THE INDICES OF SEGMENT  9 OF BOUNDARY  1
 SEGMENT TYPE    2  STRAIGHT LINE
LINE OPTION  1 USING 2 END POINTS
INPUT POINTS ARE        1.8000        .4000   AND        1.8000       2.5000
PACK OPTION=0  EVEN SPACING
INDICES ARE    1 101  81  81  FOR BOUNDARY  2
NUMBER OF SEGMENTS=  9
    1    5 ARE THE INDICES OF SEGMENT  1 OF BOUNDARY  2
SEGMENT TYPE    2  STRAIGHT LINE
LINE OPTION  1 USING 2 END POINTS
INPUT POINTS ARE        -.5000     -2.8000   AND        -.2500     -2.8000
PACK OPTION=0  EVEN SPACING
    5    9 ARE THE INDICES OF SEGMENT  2 OF BOUNDARY  2
SEGMENT TYPE    2  STRAIGHT LINE
LINE OPTION  1 USING 2 END POINTS
INPUT POINTS ARE        -.2500     -2.8000   AND         .0000     -2.8000
PACK OPTION=0  EVEN SPACING
    9   29 ARE THE INDICES OF SEGMENT  3 OF BOUNDARY  2
SEGMENT TYPE    4  CIRCLE
CIRCLE OPTION 3 USING 3 POINTS
FIRST END POINT IS        .0000      -2.8000
OTHER END POINT IS       1.4000      -2.4249
THIRD POINT ON CIRCLE IS        .7247      -2.7046
PACK OPTION=0  EVEN SPACING
NO, GOING COUNTERCLOCKWISE
   29   49 ARE THE INDICES OF SEGMENT  4 OF BOUNDARY  2
 SEGMENT TYPE    4  CIRCLE
CIRCLE OPTION 3 USING 3 POINTS
FIRST END POINT IS       1.4000      -2.4249
OTHER END POINT IS       2.4249      -1.4000
THIRD POINT ON CIRCLE IS       1.9799      -1.9799
PACK OPTION=0  EVEN SPACING
NO, GOING COUNTERCLOCKWISE
   49   59 ARE THE INDICES OF SEGMENT  5 OF BOUNDARY  2
SEGMENT TYPE    4  CIRCLE
CIRCLE OPTION 3 USING 3 POINTS
FIRST END POINT IS       2.4249      -1.4000
OTHER END POINT IS       2.7336       -.6060
THIRD POINT ON CIRCLE IS       2.6052      -1.0262
PACK OPTION=0  EVEN SPACING
NO, GOING COUNTERCLOCKWISE
   59   67 ARE THE INDICES OF SEGMENT  6 OF BOUNDARY  2
SEGMENT TYPE    4  CIRCLE
CIRCLE OPTION 3 USING 3 POINTS
FIRST END POINT IS       2.7336       -.6060
OTHER END POINT IS       2.8000        .0000
THIRD POINT ON CIRCLE IS       2.7820       -.3170
PACK OPTION=0  EVEN SPACING
NO, GOING COUNTERCLOCKWISE
   67   71 ARE THE INDICES OF SEGMENT  7 OF BOUNDARY  2
SEGMENT TYPE    2  STRAIGHT LINE
```

```
LINE OPTION  1 USING 2 END POINTS
INPUT POINTS ARE        2.8000       .0000   AND        2.8000        .2500
PACK OPTION=0  EVEN SPACING
  71  73 ARE THE INDICES OF SEGMENT  8 OF BOUNDARY  2
 EGMENT TYPE    2  STRAIGHT LINE
LINE OPTION  1 USING 2 END POINTS
INPUT POINTS ARE        2.8000       .2500   AND        2.8000        .4000
PACK OPTION=0  EVEN SPACING
  73 101 ARE THE INDICES OF SEGMENT  9 OF BOUNDARY  2
SEGMENT TYPE    2  STRAIGHT LINE
LINE OPTION  1 USING 2 END POINTS
INPUT POINTS ARE        2.8000       .4000   AND        2.8000       2.5000
PACK OPTION=0  EVEN SPACING
INDICES ARE    1   1   1  81   FOR BOUNDARY  3
NUMBER OF SEGMENTS=  1
SEGMENT TYPE    2  STRAIGHT LINE
LINE OPTION  1 USING 2 END POINTS
INPUT POINTS ARE        -.5000     -1.8000   AND        -.5000      -2.8000
PACK OPTION=7  PACKED ON BOTH ENDS USING HYPERBOLIC TANGENT STRETCHING
WITH SMALLEST INTERVAL=    .0025000
 SMALLEST INTERVAL IN SECOND SECTION =  .0025000
INDICES ARE  101 101   1  81   FOR BOUNDARY  4
NUMBER OF SEGMENTS=  1
SEGMENT TYPE    2  STRAIGHT LINE
LINE OPTION  1 USING 2 END POINTS
INPUT POINTS ARE        1.8000      2.5000   AND        2.8000       2.5000
PACK OPTION=7  PACKED ON BOTH ENDS USING HYPERBOLIC TANGENT STRETCHING
WITH SMALLEST INTERVAL=    .0025000
 SMALLEST INTERVAL IN SECOND SECTION =  .0025000
 CTIVITY OPTION  2: CREATING GRID PATCHES
INDICES ARE    1 101   1  81   FOR PATCH  1
ACTIVITY OPTION  3: CREATING AUTOPATCHES
ACTIVITY OPTION  5: PLOTTING
  1 PLOTS IN PLOT SET 1
OUTER BORDER TO BE DRAWN ON PLOT       1
  1 PATCHES TO BE DRAWN ON PLOT  1
INDICES ARE    1 101   1  81   FOR PATCH  1 ON PLOT  1
ACTIVITY OPTION  8: OUTPUTTING FILE(S)
YES, GRID IS TO BE OUTPUT
 YES,INPUTS ARE TO BE OUTPUT
```

## PART B

In order to plot on Plot3d_3.5 , the following commands are followed:

1) Access Plot3d_3.5 by inputting plot3d_3.5 at the prompt.

2) Read the file that will be the input file to Plot3d_3.5. The format to read in Plot3d_3.5 is:
re/ for or bin or unfor/x=name of the input file.
The file created by running GENIE is a binary file.
This means that using the above format, the read command will be re/bin/x=fort.47.

3) Plot3d_3.5 allows the user to set the background of the screen during plotting. The syntax for this is bg #, where #, by default represents the number representations of the different colors that are available in the software package. The different colors that are available can be found in the Plot3d_3.5 user manual. I used a white background and the command is : bg 1.

4) At the prompt, enter wa. The following sequence of commands and input follows:

| Command | Input |
|---|---|
| wa 1, grid 1 | |
| Enter I start ([,end],[,inc] ): | a |
| Enter J start ([,end],[,inc] ): | a |
| Enter K start ([,end],[,inc]): | a |
| Line Hidden, Line, or Shaded-Surface: | |
| Enter Color: | red |
| Enter Line Type: | |
| Enter Line Thickness Factor: | |
| Enter Symbol Type: | |
| Enter Symbol Size Factor: | |

By inputting a , the user is specifying to the package that all of the data in the I, J, K direction should be used. After going through these sequence, the user can plot by using the pl command. This plot will be a 2-D plot. The plot is checked in order to make sure that it conforms to  right hand co-ordinate system.. For more information on Plot3d_3.5, the user manual should be consulted.

## PART C

```
C     THIS IS A GRIDSTACK PROGRAM THAT STACKS A 2D GRID TO 3D
C
C      THE DIMENSION SIZE REFLECTS THE SIZE OF THE GRID
          DIMENSION  X1(101,81,2), X2(101,81,81,3),X3(101,81,2)
C      THE INPUT FILE TO THIS PROGRAM IS THE FORMATTED FILE THAT WAS
C       GENERATED BY PLOT3D USING THE LIST COMMAND
C
           open (5,form='formatted',file='grid3.fmt')
           open (6,form='formatted',file='outduct.fmt')
C    READ IN 2D FORMATTED, SINGLE BLOCK GRID
C
                    READ(5,*)IMAX,JMAX
                     READ (5,*) (((X1(I,J,L), I=1,IMAX), J=1,JMAX), L=1,2)
                        JMAXP1 = JMAX + 1
                       DO 100 J= 1,JMAX
                       DO 100 I=1,IMAX
                        X3(I,JMAXP1-J,1) = X1(I,J,1) + 0.5
                        X3(I,JMAXP1-J,2) = X1(I,J,2) + 2.3
100                 CONTINUE
C INITIALIZE THIRD DIMENSION MAXIMUM INDEX AND BLOCK SIZE
                     KMAX = JMAX
C STACK GRID IN THE Z-DIRECTION (K-INDEX) USING SPACING FROM Y-DIRECTION
C  (J-INDEX) AT INLET PLANE (I=1)
C
                      DO 200 K = 1,KMAX
                      DO 200 J = 1,JMAX
                      DO 200 I = 1,IMAX
                       X2(I,J,K,1) = X3(I,J,1)
                       X2(I,J,K,2) = X3(I,J,2)
                       X2(I,J,K,3) = X3(1,K,2)
200                CONTINUE
                   NGRID = 1
                     WRITE(6,*)  NGRID
                     WRITE(6,*) IMAX,JMAX,KMAX
                    DO 300 L = 1,3
          WRITE (6,*)(((X2(I,J,K,L),I=1,IMAX),J=1,JMAX),K=1,KMAX)
300            CONTINUE
                                STOP
                            END
```

## PART D

In order to get the plotting for the multigrid that is created by running the gridstack program, the following commands and input are done by the user.

| Commands | Input |
|---|---|
| Plot3d_3.5: | wa |
| wa 1, grid 1 | |
| Enter I start ([,end],[,inc]): | f |
| Enter J start ([,end],[,inc]): | a |
| Enter K start ([,end],[,inc]): | a |
| Line Hidden, Line, or Shaded-Surface: | |
| Enter Color: | red |
| Enter Line Type: | |
| Enter Symbol Type : | |
| Enter Symbol Size Factor: | |

This sequence is repeated but this time the I, J, K value will be different. Also, the color will be different. The input are :

I: Input a
J: Input f
K: Input a
Color: Input green

The sequence is repeated again. The I, J, K, and the color values are :

I: Input a
J: Input a
K: Input f
Color: Input blue

After the last sequence, the solution is plotted by using the pl command. The plot will appear on the screen. The a input represents all of the input file data in the specific direction. The f input represents the first data in the specific direction. The l input represents the last data. The different colors are used for checking purpose. We want to make sure that the I, J, K direction are in the proper direction. Also, when the plot appears on the screen, the plot is checked to make sure that it abides by the right hand co-ordinate system.

The picture of the plot is taken by entering the T command at the prompt. After about 2 minutes, the screen will start to fade. A message "Screen Image is saved in outduct-a" will appear. The outduct-a file is the image file. Plot3d_3.5, by default, creates this file anytime the user uses the T command. The user needs to quit Plot3d_3.5 in order to send the image file to the printer. The quit command for Plot3d_3.5 is : quit.

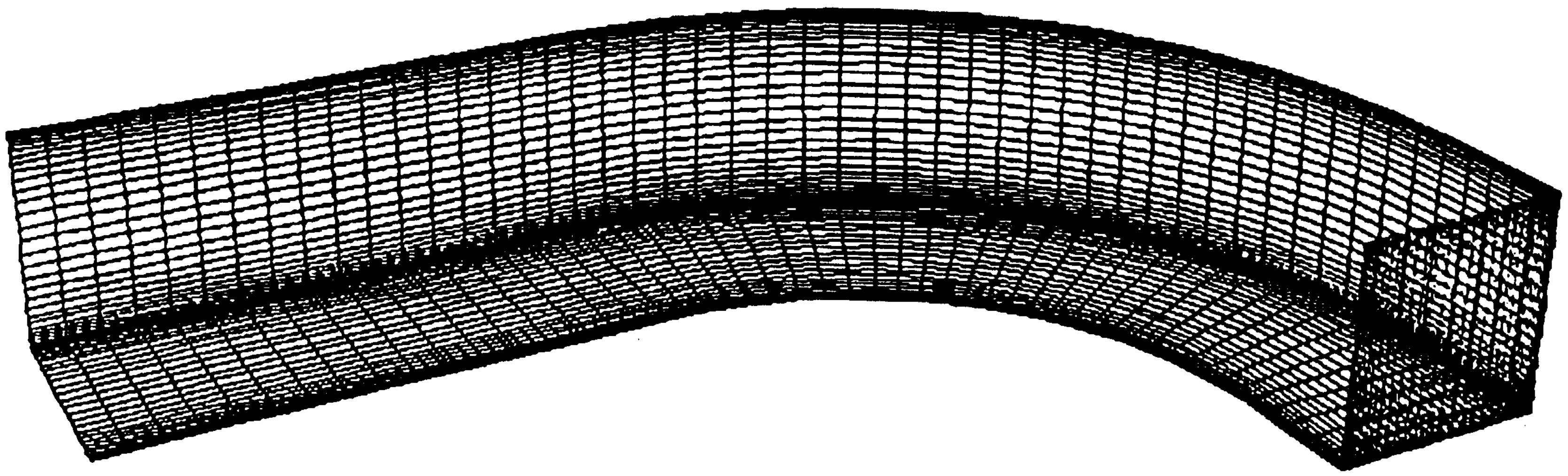
The user can also enter two lines of comment that will appear on the printout of the image file. The command for this is : t. This command is entered at the UNIX prompt not the Plot3d_3.5 prompt. After this, the image file is sent to the printer by entering the ftek command. The syntax for this command is :

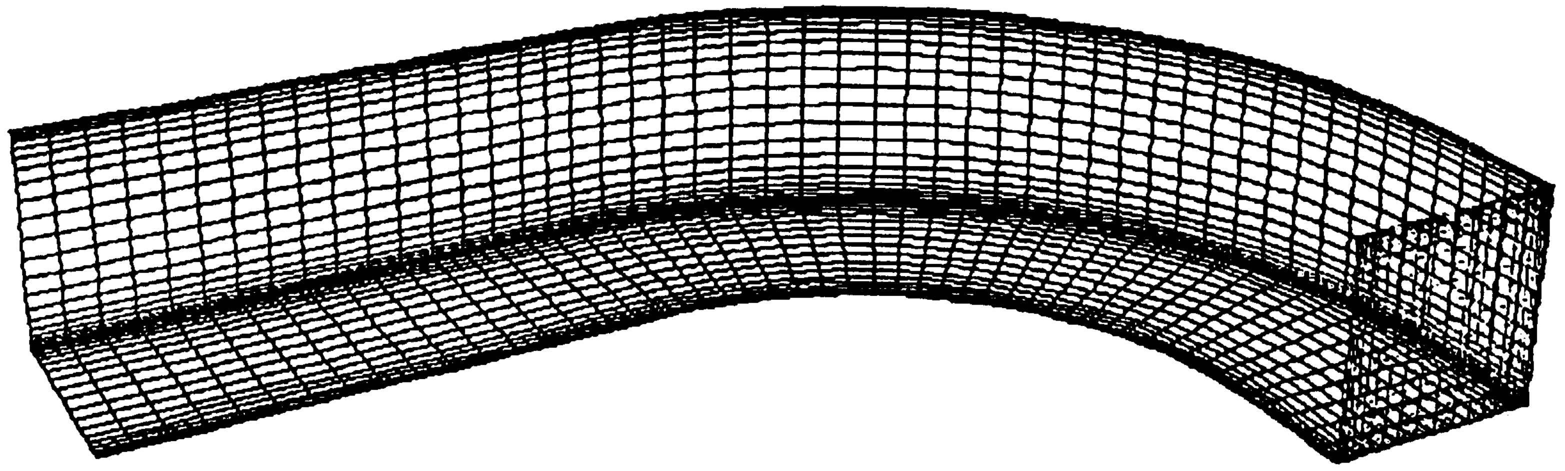
```
ftek name of image file.
```

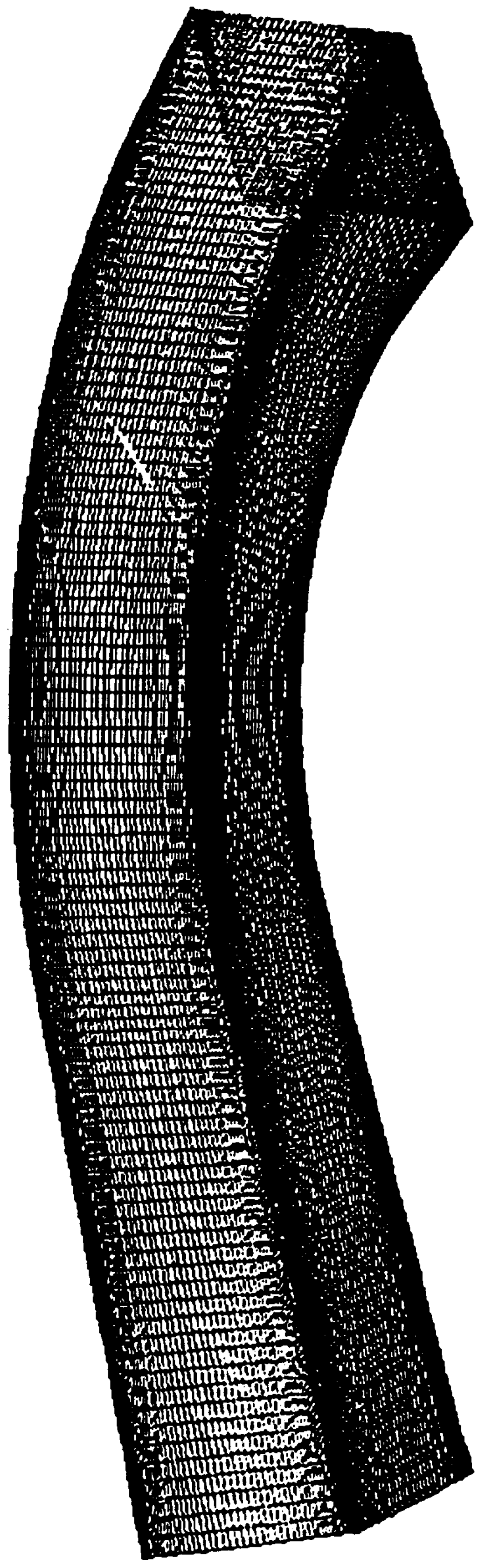
The name of the image file is the file that was created by Plot3d_3.5 when the user took a picture of the plot (outduct-a).

## PART E

BODY AND WALLS 51x31x31 GRID




BODY AND WALLS
51x21x21
GRID


BODY AND WALLS
101x61x61
GRID

## APPENDIX 2

# OPTIMIZATION OF FORTRAN PROGRAMS WITH APPLICATION TO COMPUTATIONAL FLUID DYNAMICS

## INTRODUCTION

Analysis of three-dimensional flow was thought impossible until equations governing the flow of air, water and other fluids were first published by Claude Louis M.H. Navier in 1823 and generalized by Sir George C. Stokes in 1840 (Navier-Stokes equations). The mathematics of these equations is too complex to allow an analytical solution in all but the most simple cases; thus, numerical solutions using the most powerful computers (supercomputers) currently available yield the best approximations to the solutions of the Navier-Stokes equations. Consequently, numerical analysis and supercomputers have become essential tools in Computational Fluid Dynamics (CFD) for the analysis of complex three-dimensional flows, such as the flow of fuel through the Space Shuttle Main Engine or the modeling of weather conditions around the earth.

Most computationally intense CFD code is written in FORTRAN, although most current university curricula for engineers and scientists either contain no course work in FORTRAN or contain only a single elementary course in the FORTRAN language. Agencies such as the National Aeronautical and Space Administration (NASA) and the Department of Defense (DOD) as well as industry have many large CFD codes which are often run and maintained by engineers and scientists who are lacking the experience or expertise of working with a FORTRAN code of this magnitude. Also, most have had no or very little experience with a supercomputer. As most of these users are not trained to modify FORTRAN codes to run efficiently on advanced architecture computers, they often have little knowledge or concern for the burden that these large codes place on supercomputer resources. Both the Cray X-MP at the Alabama Supercomputer Center and the Cray X-MP at NASA/Marshall Space Flight Center (MSFC) are examples of supercomputers which are used to full capacity most of the time.

## BACKGROUND

Many of the large CFD codes in current use by government and industry contain large amounts of FORTRAN written before 1980. At that time FORTRAN code was typically developed in a non-structured fashion using many language constructs which are both outdated and extremely inefficient for modern supercomputers with highly developed compilers. These compilers can overcome some of the inefficiency in the FORTRAN code but leave a great deal of room for improvement by users. On current supercomputers, some mini-supercomputers (e.g. Alliant, Convex, etc.) and even on high performance workstations (e.g. Stardent, Silicon Graphics, etc.), improvement of the FORTRAN code's efficiency can dramatically improve performance. Improved performance yields reduced demand on already saturated computer systems, reduced design time for hardware and reduced customer cost for computing resources.

The three most effective means to improve code efficiency are through vectorization, optimization and microtasking. Vectorization is a process by which a single instruction is made to perform many operations instead of just one. This process is performed by the compiler and greatly reduces the computational time of the code. Modern compilers perform much vectorization automatically; however, efficiently written code can greatly enhance the compiler's effort. Vectorization is one of the most important features of supercomputers and yields the greatest reduction in computational time required. Optimization refers to the restructuring of the code in such a way that fewer accesses from central memory need to be made and arithmetic operations are simplified, also reducing the computational time of the code. The large-scale problems requiring supercomputers for their solution are often constrained by insufficient power of two to three orders of magnitude. Modern supercomputer architectures have the potential to deliver an order of magnitude more power if used in an optimal way, thus making optimization of the code very desirable although, because of the inexperience of most code users, it is rarely done. Microtasking refers to the ability of the supercomputer to simultaneously execute segments of a program on different central processing units (CPUs). Microtasking also produces a code which allows the program to be run even when only one CPU is available. As additional CPUs become available the microtasked code has the ability to make use of the free CPU(s). Microtasking does not reduce computational time, but can reduce the wall-clock time required to run the code. This feature is extremely attractive to directors of computer centers who are attempting to make optimal use of the centers' resources. It is a relatively new feature and little effort has been made to train CFD users on its use.

## METHODS

The objective of the work was to develop written guidelines for engineers and scientists to evaluate and improve the efficiency of large scale CFD codes in current use or codes which are under development for future use. The written guidelines were to accomplish three goals:

1) Reduce the number of man-hours required by the user who is responsible for improving the efficiency of a code;

2) Minimize the amount of control processing unit (CPU) time the code requires to run;

3) Optimize the use of multiple CPUs available on the supercomputer (parallel processing).

The accomplishment of the objective involved identification of software tools currently in existence and development of guidelines to aid in the evaluation and restructuring of the FORTRAN code for the user. Cray Research Incorporated (CRI) is currently the only manufacturer of supercomputers in the United States; therefore, all very large CFD codes are developed to be run on a Cray computer. Thus, the first step toward accomplishing the objective was to incorporate the software development tools available from CRI into the written guidelines of this project. The current tools available from CRI (FLOWTRACE, LOOPMARK and Autotasking) are not readily used by scientists and engineers since they require study and experience beyond the daily scope of work of the typical user. The written guidelines include instructions on how and when these tools can be used to the greatest benefit.

Before attempting code optimization, it was important to identify the most time-consuming portions of code and then concentrate the efforts on these. CRI offers a tool called FLOWTRACE, which generates run-time statistics. Output from FLOWTRACE indicates the total computing time used by the program, the amount of time spent in each subroutine, the percentage of total time spent in each subroutine, the number of times each subroutine is called, and the average time per call. It also produces a calling tree which identifies the calls made at each level of the program. FLOWTRACE was used to determine which sections of the code should be investigated for possible improvement so that time would not be spent improving a section of code which had little impact on overall performance.

Another tool available on the CRI line of supercomputers is called LOOPMARK. LOOPMARK examines each inner DO loop in the FORTRAN code and comments as to whether it is automatically vectorized by the compiler; it also prints an explanation for each loop that is not vectorized. LOOPMARK was used to determine which loops could be restructured to allow automatic vectorization by the compiler.

Autotasking is a tool that is currently only available through use of the cf77 compiler. It first vectorizes the inner loop, then breaks up the outer loop, determining which portions can be run in parallel, which variables are private and which are shared, and whether the loop performs enough work to warrant the overhead incurred by using more than one processor. The additional overhead associated with Autotasking is incurred even if only one CPU is utilized; therefore invoking Autotasking when all CPUs are in use can actually result in an increase in CPU time with no decrease in wall-clock time. At least 65-70% of the code must be able to be run parallel for Autotasking to be beneficial in any case. Because the program statements are no longer executed in deterministic order, Autotasking can also produce results different from that of the original code. The user must determine whether these differences are significant.

## OUTCOME OF TASKS

The outcome of this work is a set of written guidelines which will guide the code user through the efficiency evaluation of any CFD FORTRAN code. Thorough understanding of the mathematics and physics in the code will not be necessary to carry out an analysis of the efficiency of the code. Once the evaluation is done, more guidelines are available to lead the engineer or scientist through an evaluation and modification of the code's level of vectorization and optimization and the use of autotasking. This "tool box" of improvements provides easy to understand and easy to implement modifications to CFD codes which engineers and scientists can use to improve a code without having to know all the details of efficient FORTRAN programming. This set of guidelines will aid any engineering group whose main function is code usage. New candidate codes can be immediately evaluated to determine their efficiency with respect to other codes in current use. Codes selected for use by the engineering group can then be taken through an evaluation/modification phase resulting in less demand on computing resources.

Two tools, LOOPMARK and FLOWTRACE, are available from Cray Research to assist you in evaluating the efficiency of your FORTRAN code. LOOPMARK is run at compile time; therefore no additional overhead is incurred by using it. FLOWTRACE executes at run time, and thus requires more CPU time than running the code without it; however, it provides valuable information that can save you a great deal of time when trying to improve the code's efficiency. LOOPMARK and FLOWTRACE can be used either separately or together.

1. Compile the code using LOOPMARK. LOOPMARK produces a listing file with each DO loop marked as either scalar or vectorized. Except for outer loops, which cannot be vectorized, LOOPMARK lists the reasons that a loop is not vectorized. To compile your code using LOOPMARK, use either

```
cft -v msgs filename.f
```

or

```
cft77 -em filename.f
```

or

```
cf77 -em filename.f
```

The following is an example of output from LOOPMARK:

ORIGINAL PAGE IS OF POOR QUALITY

```
175     175. S-------<        DO 50  K=1,KEND(NT)
176     176. :    V---<       DO 50  I=1,IMXI(NT)
177     177. :    V           QW(I,K)=HF
178     178. :---V--->     50 CONTINUE
```

```
                                   V E C T O R I Z A T I O N   I N F O R M A T I O N
                                   -------------------------------------------------
*** *** Loop starting at line 124 was vectorized
*** *** Loop starting at line 128 was vectorized
*** *** Loop starting at line 133 was vectorized
*** *** Loop starting at line 139 was not vectorized because
        the loop contains input/output operations
*** *** Loop starting at line 144 was not vectorized because
        the loop contains input/output operations
*** *** Loop starting at line 151 was not vectorized because
        the loop contains input/output operations
*** *** Loop starting at line 160 was not vectorized because
        the loop contains input/output operations
*** *** Loop starting at line 164 was not vectorized because
        the loop contains input/output operations
*** *** Loop starting at line 169 was not vectorized because
        the loop contains input/output operations
*** *** Loop starting at line 176 was vectorized
*** *** Loop starting at line 233 was not vectorized because
        the loop contains a scalar store on NTOTAL
```

2. Run the code using FLOWTRACE to determine in which subroutines the most effort should be concentrated. Output from FLOWTRACE indicates the total computing time used by the program broken down into the time spent in each subroutine and the percentage of total time spent in each. It also indicates the number of times each subroutine is called and the average time per call, and produces a calling tree which identifies the calling routine for each. When attempting to improve the code's efficiency, you should first examine those subroutines in which the largest percentage of the total time is spent. You should also examine any subroutine that is called a large number of times. Although the percentage of total time spent in it may be small, improving a subroutine that is called many times can decrease the total CPU time required to run the code. FLOWTRACE can be invoked using either the cft, cft77 or cf77 FORTRAN compiler by using the following compile commands:

```
cft -e f filename.f

cft77 -e f filename.f

cf77 -F
```

The following is an example of output from FLOWTRACE:

```
1 F L O W T R A C E  --  Alphabetized summary
0     Routine              Time executing   Called   Average T
   24 BTRI            6.832   ( 14.40%)      510       0.013 Called by     LHSSI    LHSSO
      @00062361a                                                             315      195
   10 CONTRL          0.438   (  0.92%)        5       0.088 @00027747a Called by    ROTOR
   43 CONVRG          0.245   (  0.52%)        5       0.049 @00031204a Called by   CONTRL
   39 CORREC          0.419   (  0.88%)        5       0.084 @00031526a Called by   CONTRL
   20 FLUXR           3.300   (  6.95%)    37360           > Called by     RHSSI    RHSSO
      @00047205a                                                           23395    13965
    8 GETAJA              >   (  0.00%)       13           > Called by    CONTRL   REDWRT
      @00111741a                                                              10        3
   14 GETOLD              >   (  0.00%)       20           > Called by    CONTRL   CONVRG
      @00111631a                                                              10       10
    7 GETQ            0.002   (  0.01%)       70           > Called by    CONTRL   CONVRG    CORREC
      @00111565a                                                              20       10        30
   33 GETTMP              >   (  0.00%)        5           > @00111675a Called by    RHSSO
    9 GETXYZ          0.004   (  0.01%)       40           > Called by    CONTRL   CORREC    INEXS
      @00112004a                                                              10       25         2
   46 INEXS           0.005   (  0.01%)        1       0.005 @00026422a Called by    ROTOR
   22 LHSSI           3.190   (  6.72%)        5       0.638 @00047747a Called by   CONTRL
   35 LHSSO           1.755   (  3.70%)        5       0.351 @00054126a Called by   CONTRL
   18 MUKN            0.288   (  0.61%)       10       0.029 Called by     RHSSI    RHSSO
      @00047027a                                                               5        5
   19 MUTRSI          3.439   (  7.25%)        5       0.688 @00063470a Called by    RHSSI
   34 MUTRSO          1.070   (  2.25%)        5       0.214 @00077761a Called by    RHSSO
    2 OPNCLO          0.009   (  0.02%)        1       0.009 @00111164a Called by    ROTOR
   49 OUTPUT          0.723   (  1.52%)        1       0.723 @00026072a Called by    ROTOR
    6 PUTAJA              >   (  0.00%)        3           > @00111372a Called by   REDWRT
   13 PUTOLD              >   (  0.00%)       10           > @00111262a Called by   CONTRL
    5 PUTQ            0.001   (  0.00%)       38           > Called by    CONTRL   CORREC    REDWRT
      @00111216a                                                              10       25         3
   30 PUTTMP              >   (  0.00%)        5           > @00111326a Called by    RHSSO
    4 PUTXYZ              >   (  0.00%)        3           > @00111435a Called by   REDWRT
    3 REDWRT         16.794   ( 35.39%)        2       8.397 @00106441a Called by    ROTOR
   17 RHSSI           1.082   (  2.28%)        5       0.216 @00035550a Called by   CONTRL
   26 RHSSO           0.840   (  1.77%)        5       0.168 @00040363a Called by   CONTRL
    1 ROTOR           0.003   (  0.01%)        1       0.003 @00025203a Called by
   23 SMATRX          5.092   ( 10.73%)    37360           > Called by     LHSSI    LHSSO
      @00061121a                                                           23395    13965
   31 SRINT           0.016   (  0.03%)        5       0.003 @00045133a Called by    RHSSO
   21 VFLUX           0.921   (  1.94%)    26090           > Called by     RHSSI    RHSSO
      @00046571a                                                           21210     4880
   25 VMAT            0.974   (  2.05%)    26090           > Called by     LHSSI    LHSSO
      @00060621a                                                           21210     4880
 * * * TOTAL         47.446      127683 Total calls
1 F L O W T R A C E  --  Calling tree
         1     ROTOR     00025203a
         2        OPNCLO    00111164a
         3        REDWRT    00106441a
         4           PUTXYZ    00111435a
         5           PUTQ      00111216a
         6           PUTAJA    00111372a
         7           GETQ      00111565a
         8           GETAJA    00111741a
         9           GETXYZ    00112004a
        10        CONTRL    00027747a
        11           PUTQ      00111216a
        12           GETQ      00111565a
        13           PUTOLD    00111262a
        14           GETOLD    00111631a
```

```
15        GETAJA    00111741a
16        GETXYZ    00112004a
17        RHSSI     00035550a
18          MUKN      00047027a
19          MUTRSI    00063470a
20          FLUXR     00047205a
21          VFLUX     00046571a
22        LHSSI     00047747a
23          SMATRX    00061121a
24          BTRI      00062361a
25          VMAT      00060621a
26        RHSSO     00040363a
27          MUKN      00047027a
28          FLUXR     00047205a
29          VFLUX     00046571a
30          PUTTMP    00111326a
31          SRINT     00045133a
32            GETQ      00111565a
33          GETTMP    00111675a
34          MUTRSO    00077761a
35        LHSSO     00054126a
36          SMATRX    00061121a
37          BTRI      00062361a
38          VMAT      00060621a
39        CORREC    00031526a
40          PUTQ      00111216a
41          GETQ      00111565a
42          GETXYZ    00112004a
43        CONVRG    00031204a
44          GETQ      00111565a
45          GETOLD    00111631a
46      INEXS     00026422a
47        GETQ      00111565a
48        GETXYZ    00112004a
49      OUTPUT    00026072a
```

3. LOOPMARK and FLOWTRACE can be invoked simultaneously by using the commands:

```
cft -v msgs filename.f

cft 77 -e fm filename.f

cf77 -F -em filename.f
```

NOTE: It is advisable to run the code after any major changes are made to insure that the output is the same as that of the original code. If several changes are made before the code is run, it can become extremely difficult to determine which change caused the discrepancy in the output.

## GUIDELINES FOR VECTORIZATION

1. A loop containing another loop will never vectorize. Because the innermost loops can be vectorized, the efficiency of the code can be greatly improved by ensuring that the innermost loop is executed for the largest number of iterations. A section of code containing nested loops with no statements between.LS1

   Example:
   ```
   DO 10 I = 1, L
      DO 20 J = 1, M
         DO 30 K = 1, N
   ```

   can always be reordered with no unexpected effect. If other statements occur within the nested loops

   Example:
   ```
   DO 10 I = 1 , L
      statements
      DO 20 J = 1, M
         statements
         DO 30 K = 1, N
   ```

   the loops can still be reordered, but you must be careful to move any statements in between that may change what is happening in the code.

2. An outer loop which is executed for four or fewer iterations, even if vectorized by the compiler, can be "unrolled", which may significantly decrease the amount of CPU time used. (This is somewhat machine dependent.) The term "unrolling" means to repeat that section of the code for each iteration, putting in constants for the array subscripts instead of variable names.

   Example:
   ```
             DO 10 J = 1, 2
                DO 10 K = 1, 1000
       10    A(J,K) = expression
   ```

   can be unrolled to produce

   ```
             DO 10 K = 1, 1000
             A(1,K) = expression
       10    A(2,K) = expression
   ```

3. A loop with only two iterations will not be vectorized by the compiler and should always be unrolled.

4. An inner loop with few iterations may be marked by LOOPMARK as a "Vs" (short vector) loop. In this case, no benefit can be produced by unrolling the loop.

5. Any loop that contains a READ or WRITE statement will not vectorize. A WRITE statement within an IF statement, such as

```
IF DEBUG THEN
   WRITE (6, 1000)
```

will inhibit vectorization even if the value of DEBUG is set to false. If these I/O statements can be omitted or moved outside the loop (for example, reading in all the elements of an array in a separate loop before the loop that uses it), vectorization can be achieved.

6. A loop that contains a call to a subroutine or external user-defined function will not vectorize. Most intrinsic functions (SQRT, SIN, etc.) are vectorized, and loops containing calls to them are vectorizable. Some ways to force vectorization are:

   a. Use a statement function instead of an external function call. (NOTE: Be sure to look at the calling tree produced by FLOWTRACE to determine which other subroutines call this function if you decide to eliminate the function altogether.)

   b. Move the entire subroutine code into the loop. (NOTE: Be sure to look at the calling tree produced by FLOWTRACE to determine which other subroutines call this subroutine if you decide to eliminate the subroutine altogether.)

   c. Move the loop into the subroutine and make a call to it from the original subroutine.

      This requires a more thorough understanding of the code, and can only be done if the called subroutine is not called from any other part of the program.

   d. Move the call outside the main loop and into a separate loop of its own. This can only be done if the following conditions are met:

      a. The called subroutine does not make assignments to variables referenced in the loop.

      b. There are no STOP or alternate RETURN statements in the subroutine.

      c. The subroutine arguments are not arrays.

      d. Any subroutine called from within this subroutine meets these same conditions.

7. A loop containing a backward GO TO will not vectorize unless it is restructured. Restructuring requires analysis of the loop and is sometimes impossible. When it is possible, it requires that the loop be rewritten with a forward GO TO or a jump out of the loop when a desired value is reached (see

#9).

8. A loop containing an assigned GO TO will not vectorize and cannot be restructured without rewriting to eliminate the ASSIGN statement, which may not be possible without a thorough understanding of the code.

9. A loop containing a jump out of the loop based on an IF statement will not vectorize. This can sometimes be rewritten so that the rest of the loop is only executed if the opposite of the IF condition is true. (NOTE: This does not always speed up execution.)

Example:
```
          DO 10 I = 1, N
             statements
             IF (A .GE. B) GO TO 10
             statements
   10     CONTINUE
```

can be rewritten as

```
          DO 10 I = 1, N
             statements
             IF (A .LT. B) THEN
                statements
             ENDIF
   10     CONTINUE
```

In cases where this simple technique will not work, there is no easy way to restructure the loop without a complete understanding of the code and a thorough knowledge of FORTRAN. Those interested should consult the discussion of stripmining in A Guidebook to FORTRAN on Supercomputers by John M. Levesque and Joel W. Williamson.

10. A loop containing two IF statements with mutually exclusive conditions that have scalar variables being updated depending on which condition is true will be flagged by LOOPMARK as non-vectorizable because "scalar values are updated more than once". The loop will vectorize if it is rewritten using an IF-THEN-ELSE structure.

Example:
```
          DO 10 I = 1, N
             statements
             IF (A .NE. B) THEN
                statement set 1
             ENDIF
             IF (A .EQ. B) THEN
                statement set 2
             ENDIF
   10     CONTINUE
```

can be rewritten as

```
          DO 10 I = 1, N
             IF (A .NE. B) THEN
                statement set 1
             ELSE
                statement set 2
             ENDIF
     10   CONTINUE
```

11. A loop containing recursion on an array or scalar will not vectorize. This includes using a scalar or array constant on the right hand side of an equation before using it on the left hand side, and assigning an array element a value that is the result of a calculation involving another element of the array with a subscript decremented by a constant or an expression.

    Example:

```
          DO 100 I = 2, N
    100     A(I) = A(I - 1) + expression
```

    In this case, LOOPMARK will flag the loop with "a recurrence was found on A". This can be resolved by storing a copy of A in a temporary array.

```
          DO 100 I = 1, N
    100      TEMP(I) = A(I)
          DO 200 I = 2, N
    200      A(I) = TEMP(I - 1) + expression
```

    Although this permits vectorization of the loop, it may not be feasible because of the additional memory requirement.

1. If the compiler can recognize invariant code within the loop, it can pre-compute it before the loop and store the result in a register, which can greatly reduce CPU time. Placing the invariant portion of code inside parentheses assists the compiler in the recognition process.

   Example:
   ```
          DO 10 I = 1, N
   10        A(I) = X + B(I) + Y
   ```

   is optimally written as

   ```
          DO 10 I = 1, N
   10        A(I) = (X + Y) + B(I)
   ```

   (NOTE. With the capabilities of current compilers, this is more efficient than

   ```
          TEMP = X + Y
          DO 10 I = 1, N
   10        A(I) = B(I) + TEMP
   ```
   )

2. Subexpressions common to two or more expressions can be computed once and stored in registers if the compiler is able to recognize them. The compiler's effort can be enhanced by placing the subexpression in parentheses

   Example:
   ```
          A(I) = B(I) + (C(I) * D(I))
          E(I) = F(I) + G(I)
          H(I) = P(I) + (C(I) * D(I)) / Q(I)
   ```

   Again, this is more efficient than placing the common subexpression in a temporary variable and using the temporary in the computations.

3. If two short loops contain common subexpressions and the loops can be combined without changing the meaning of the code, do so. This allows better utilization of the compiler's optimization capabilities.

4. Multiplication operations are less computationally expensive than are division operations. Change division by a constant to multiplication whenever possible without losing accuracy

   Example:
   ```
          A(I) = B(I) / 2
   ```

   becomes
   ```
          A(I) = B(I) * 0.5
   ```

5. Convert floating point exponentiation    integer exponentiation whenever possible.

Example:

```
X ** 4.0
```

becomes

```
X ** 4
```

Microtasking can be accomplished automatically throughout the code using autotasking or selectively by including compiler directives in the specific sections where microtasking is desired. Use of compiler directives to force microtasking requires that the user be thoroughly familiar with the code. He must understand the scope of the variables affected by the code section and understand the effects of simultaneous execution on different CPUs. CRI provides tools such as SPY and Perftrace to aid in the determination of scope. Those interested in using compiler directives to enforce microtasking should refer to the CRI User's Guide to cf77.

Autotasking may be invoked using the command

```
cf77 -ZP
```

and may also be used in combination with LOOPMARK and/or FLOWTRACE with the commands

```
cf77 -em -Zp        (LOOPMARK)

cf77 -F -Zp         (FLOWTRACE)

cf77 -em -F -Zp          (both)
```

Autotasking does not always reduce the wall-clock time required to run the code because it depends on the number of CPUs available, and can only reduce the time by 1/n, where n is the total number of CPUs, at best. Autotasking is more likely to be useful when running the code at off-peak times.

Be sure to examine all output after running the code with autotasking, as statements may be executed in a different order and may produce different results.

## RESULTS

These methods and techniques were applied to a version of a large CFD code called ROTOR, which was obtained from NASA/Marshall Space Flight Center. This version of ROTOR contained 31 routines, including the main program. Of these 31, there were 13 which were called relatively few times and in which 0.01% or less of the total CPU time was spent. Of the remaining 18 subroutines, four were completely vectorized, including the two in which the largest percentages of time were spent, and two were input/output routines. Therefore, efforts to enhance vectorization were primarily limited to 12 of the subroutines.

Vectorization enhancement efforts in three of these subroutines resulted in an increase in the amount of CPU time; the largest increase was 8.988 units, with a mean increase of 3.077 units. The unit time decrease observed in the other subroutines ranged from 0.082 to 58.771, with a mean decrease of 11.422 units. When efforts in these subroutines were exhausted, the 13 "insignificant" subroutines were also examined and modified. After all modifications had been made, the total time for execution of the program dropped from 1058.703 to 961.288 units, a decrease of 97.415 units, or 9.2%.

Examination of each subroutine revealed that no code optimization could be performed.

The code was compiled and run with the autotasking feature enabled. The value for output parameter DRIMAX differed from the output of the original code in the ninth decimal place, and that for DROMAX in the sixth decimal place. Inlet and exit condition results differed from those obtained from the original version of the code as follows

| Parameter | Decimal place |
|---|---|
| ROINL | 12 |
| RUINL | 9 |
| RVINL | 22 |
| RWINL | 24 |
| PSINL | 12 |
| PTINL | no variation |
| ROEXT | 8 |
| PSEXT | 9 |
| PTEXT | 9 |
| Velocity | 8 |

The user must determine whether these differences are significant.

ORIGINAL PAGE IS
OF POOR QUALITY

## APPENDIX 3

## Implementation of Equilibrium Chemistry

### 1. Introduction

The knowledge of chemical equilibrium compositions of a chemical system permits one to calculate theoretical thermodynamic properties for the system. These properties can be applied to a wide variety of problems in chemistry and propulsion engineering. Some applications are the design and analysis of equilibrium such as compressors, turbines, nozzles, engines, shock tubes, heat exchangers, and processing equipment.

Considerable numerical calculations are necessary to obtain equilibrium compositions for complex chemical systems. This has resulted in a number of digital computer programs to do the calculations. A computer program (CEC, 1) written at NASA Lewis Research Center in 1961-1962 has had a wide acceptance. However, in calculating the fast chemical reactions in large-scale and multidimensional problems, this general chemistry code (CEC) is so inefficient. There are several requirements of an equilibrium chemistry solver to be implemented in CFD codes. These are the computational efficiency and the reliability. The main objective of the present study is to implement and modify the existing equilibrium chemistry solver to meet these requirements.

To improve the reliability, the equilibrium solver with the free energy minimization procedure is selected. Chemical equilibrium is usually described by either of two equivalent formulations - equilibrium constants or minimization of free energy. In comparison with Gibb's free energy minimization method, the equilibrium constant method has several disadvantages which are more bookkeeping, numerical difficulties with use of chemical components, and more difficulty in testing for presence of some condensed species. Furthermore, the Gibbs free energy is most easily minimized. For these reasons, the free-energy minimization formulation is used.

The primary requirement of an equilibrium solver in a large finite-difference code is that it be fast. A typical run of multidimensional transient spray reacting flows requires the equilibrium problem to be solved about a million times. Typically, the speed of an algorithm in solving a single problem is measured in milliseconds. Each extra millisecond required by the solver will translate into about 17 additional minutes of computer CPU time. To improve the efficiency in terms of computing time and convergence, the present equilibrium chemistry solver will adopt a hierarchical algorithm [2] which combine the speed of Gauss-Siedel method and the convergence of Newton method.

## 2. General Program Features

The present equilibrium code(ADAPT) has the functionally similar features with other existing equilibrium solvers except the adaptability with the multidimensional CFD codes. ADAPT solution procedure using Gibbs' free energy minimization technique provides useful options either a complete-combustion equilibrium model or a full equilibrium model, in which the full array of species existing under chemical equilibrium conditions is obtained at each grid point with the given pressure and enthalpy.

In general, the ADAPT code provides a computer-oriented mathematical model of the ramjet combustor and nozzle which can be used to estimate the performance of a given ramjet design and to parametrically evaluate the effects of changes in the design on combustor and nozzle performance. It is written with numerous user options to allow consideration of a number of potential combustor and nozzle designs, including provision for fuel injection in liquid from the walls or instream fuel injectors. Multiple fuel injector locations may be incorporated.

When ADAPT is implemented with CFD codes, the basic solution procedure are summarized as follows:

a) evaluating initial profile data for the dependent variables for which data have been input. Compute element mass fractions from the species mass fractions input. Extract from memory the necessary thermodynamic data for each of the species included. In general, initialize the calculation.

b) perform equilibrium calculation at given enthalpy, pressure, and element mass fractions to obtain species.

c) obtain all other profile information, e.g. density, specific heat, physical coordinates, viscosity, etc., from CFD codes.

d) determine the change in the dependent variables due to convection and turbulent diffusive forces through mean flow solution of the governing equations by using CFD codes.

e) return to step (b).

## 3. Validation of Present Equilibrium Chemistry Model

To validate the present equilibrium chemistry model, the point chemistry results of ADAPT are compared with CEC results. Propane ($C_3H_8$) is selected as a test fuel. 14 species are considered in ADAPT while 58 species are considered in CEC. The selected condition is stoichiometric. The numerical results of ADAPT and CEC are summarized in Table 1 and Table 2. The flame temperature and the mole fraction of the dominant species ($CO_2$, $H_2O$) are the key variables to evaluate. As shown in Tables 1 and

2, overall agreement for flame temperature and species mole fractions are satisfactory. In P/H case listed in Table 1, the deviations in flame temperature and species mole fractions are within 0.9 % and 0.3 %, respectively. In T/P case listed in Table 2, the deviations of species mole fractions are within 0.3 %. In terms of CPU time listed in Table 1 and 2, ADAPT is computationally more efficient than CEC. Especially for T/P case, ADAPT is roughly four times faster than CEC. These numerical results indicate that ADAPT can handle the complex equilibrium chemistry problems with the limited participating chemical species and the computational efficiency.

## 4. Conclusions

Numerical results of the present equilibrium chemistry code have a satisfactory agreement with CEC results. These numerical results indicate that the present equilibrium solver can handle the complex equilibrium chemistry problems with the limited participating chemical species and the computational efficiency. The future works include the implementation of ADAPT equilibrium code in CFD codes such as FDNS and MAST.

## References

1. Gordon, S. and McBride, B., "Computer Program for Calculation of Complex Chemical Equilibrium Compositions, Rocket Performance, incident and Reflected Shocks, and Chapman-Jouguet Detonations". NASA-273, 1976.

2. Meintjes, K. and Morgan, A., "Performance of Algorithms for Calculating the Equilibrium Composition of a Mixture of Gases.", Journal of Computational Physics, Vol. 60, pp. 219-234, 1985.

**Table 1.** Comparison of CPU time, predicted flame temperature and species mole fraction for P/H condition

| | CRAY-XMP CPU time (sec) | Flame temp.($^oK$) | Mole Fraction $H_2O$ | Mole Fraction $CO_2$ |
|---|---|---|---|---|
| Present Code | 0.059 | 2219.8 | 0.14298 | 0.10057 |
| CEC Code | 0.070 | 2218.0 | 0.14295 | 0.10037 |

**Table 2.** Comparison of CPU time, predicted flame temperature and species mole fraction for T/P condition

| | CRAY-XMP CPU time (sec) | Flame temp.($^oK$) | Mole Fraction $H_2O$ | Mole Fraction $CO_2$ |
|---|---|---|---|---|
| Present Code | 0.016 | 2218.0 | 0.14299 | 0.10058 |
| CEC Code | 0.066 | 2218.0 | 0.14296 | 0.10038 |