Supersonic Civil Airplane Study and Design: Performance and Sonic Boom

Samson Cheung

January 1995
NCC2-617

Surface Grid Generator

Upwind Parabolized Navier-Stokes code
(finite volume scheme)

Isentropic Euler
Axisymmetric code
(method of characteristics)

L/D optimized?

OPTIMIZER

CFD Calculation


Unclas
Supersonic Civil Airplane Study and Design: Performance and Sonic Boom

Samson Cheung

This final report summarizes the work performed from July 1989 to Jan. 1995. The work is supported by NASA Co-operative Agreement NCC2-617. This report consists of four parts. The first part is the introduction of the research effort. The second part describes the work and results from July 1989 to June 1993. The third part describes the work and results from July 1993 to January 1995. A summary is given at the end of this report.

1 INTRODUCTION

The present supersonic civil airplane, the Concorde, is a technological break-through in aviation history. However, it is an economical disaster for two main reasons. The first is her low aerodynamic performance, that allows only 100 passengers to be carried for a short-range flight with expansive airfare. Another reason is that the shock waves, generated at supersonic cruise, coalesce and form a classical N-wave on the ground, forming a double bang noise termed sonic boom, which is environmentally unacceptable. To enhance the U.S. market share in supersonic civil transport, an airframer's market risk for a low-boom airplane has to be reduced.

Since aircraft configuration plays an important role on aerodynamic performance and sonic boom shape, the configuration of the next generation supersonic civil transport has to be tailored to meet high aerodynamic performance and low sonic boom requirements. Computational fluid dynamics (CFD) can used to design airplanes to meet these dual objectives. The work and results in this report are used to support NASA's High Speed Research Program (HSRP).

In this five years of study and research, CFD tools and techniques have been developed for general usages of sonic boom propagation study and aerodynamic design. In the beginning of the 90's, sonic boom extrapolation technique was still relied on the linear theory developed in the 60's for the nonlinear techniques were computationally expensive. A fast and accurate sonic boom extrapolation methodology (Section 3.2), solving the Euler equations for axisymmetric flow, has brought the sonic boom extrapolation technique up to the 90's standard.

Parallel to the research effort on sonic boom extrapolation, CFD flow solvers have been coupled with a numeric optimization tool to form a design package for aircraft configura-
tion. This CFD optimization package has been applied to configuration design on a low-boom concept (Section 2.3) and an Oblique All-Wing concept (Section 2.4) prior to the wind-tunnel models are built and tested at Ames. The tunnel test results have validated the CFD technique and design tools.

Moving to the world of parallel computing, the aerospace industry needs a numeric optimization tool suitable for parallel computers. A nonlinear unconstrained optimizer for Parallel Virtual Machine has been developed for aerodynamic design and study. Study in Section 3.3 demonstrates the capability of this optimizer on aerodynamic design.

2 PREVIOUS WORK/RESULTS

The work and results described in this section was begun in July 1989. The first project was to use CFD tools and existing linear theory to predict waveform signatures at some distances from flight vehicles. The aim of this study was to demonstrate and develop the technique of sonic boom prediction by CFD. The next step was to apply this developed technique to low-boom configurations.

The second project, which was the continuation of the first one, was to develop a CFD optimization package for design process on meeting the dual objectives of high aerodynamic performance and low sonic boom loudness. This optimization package was applied to three different High Speed Civil Transport (HSCT) baseline configurations and a generic body of revolution.

A wind-tunnel model (Ames Model 3) was built based on one of the modified HSCT baseline configuration. This model was tested in June 1993. The test results were used to validate the design method. Publication of the result was limited due to the sensitive nature of the project.

A counterpart of the conventional HSCT concept was the Oblique All-Wing (OAW) concept. CFD computational supports, as well as optimization calculations, were provided to the OAW design team consisting personnals from NASA Ames Research Center, industry, and university. The aim of the project was to design a realistic configuration for wind-tunnel test. The model was built and tested at Ames in June 1994.

2.1 Sonic Boom Prediction Technique

In the early stage of sonic boom prediction activity, two major things were involved. The very first thing was to identify the capability of CFD in sonic boom prediction. The second thing was to apply these CFD tools to predict sonic boom signals of varies configurations after necessary code modification, grid refinement study, and comparison with supersonic linear theory.
2.1.1 Method Validation

A three-dimensional parabolized Navier-Stokes code, UPS3D, developed at Ames was used as the flow-solver. It is a space-marching code with finite-volume approach. The near field solution of a simple wing/body configuration was calculated by UPS3D, and the overpressure signal at some desired distances were obtained either by the axisymmetric option of UPS3D or a quasi-linear extrapolation code, based on Whitham's F-function theory. Later I realized that using Lighthill integral to calculate the F-function for non-axisymmetric aircraft was more accurate, I wrote a Fortran code, LHF, for sonic boom prediction based on Lighthill integral. This code is available from Ames Software Library. A copy of LHF is attached in Appendix A. The figure below is a brief summary of the sonic boom extrapolation process.

A series of studies on grid refinement, including solution adaptive grid, and on sensitivity of initial distance of extrapolation were conducted. It was found that viscous calculation was unnecessary for sonic boom prediction. However, the grid must be sufficiently fine in the regions of shock and expansion waves. In order to capture all the nonlinear effects in a three-dimensional flow, the near-field overpressure should be captured at about one span length below the flight track before extrapolating to the far field. The detail results were published in AIAA Journal of Aircraft and NASA Technical note.

In summary, the tools for sonic boom prediction had been identify and validated in the above study. The combination of CFD and Whitham's method gave a relative efficient tools for sonic boom prediction. Nevertheless, the CFD codes was still computationally expensive for design optimization runs.
2.1.2 Boom Prediction for Low-Boom Configurations

With the experience on grid refinement study and the extrapolation procedure, the prediction tools were being used to predict the sonic boom of two low-boom configurations designed by Boeing aircraft company and Langley research center.

Each of the two configurations consisted of two separated parts, namely, the wing and the fuselage. The wing was defined by data in spanwise cuts, whereas the fuselage was defined by data in streamwise cuts. In order to create a single wing-fuselage surface grid for UPS3D code, a grid generator (SAMGRID) was written to defined the wing in streamwise cuts and aggregated the wing to the fuselage. Computation results of the two configurations are shown below.

![Langley's Mach 2 Configuration](image1)

![Boeing's Low-Boom Configuration](image2)

The sonic boom signals calculated from the CFD prediction tools were compared to the wind-tunnel data of the Langley’s configuration. The computational results of the Boeing’s configuration was used to validate the linear design method used by Boeing.

2.2 Supersonic Airplane Design

The need for simultaneous sonic boom and aerodynamic optimization was highlighted when it became clear that designed to a strict sonic boom constraint suffered an unacceptable performance penalty. Therefore, low-boom design studies must carefully balance the trade-off between sonic boom loudness and aerodynamic performance. A CFD optimization package was developed to demonstrate the methodology for the optimization of supersonic airplane designs to meet the dual objectives of low sonic boom and high aerodynamic performance.

In this project, an optimizer with linear and nonlinear constraints was first identified, and then an efficient CFD flow solver was chosen. This CFD code had to be sufficiently fast because more than 90% of the computational time were used in CFD calculations. Before this optimization was used to design low-boom wind-tunnel model (Section 2.3), it was tested and exercised by improving aerodynamic performance of a low-boom wing/body configuration and a body of revolution.
2.2.1 CFD Optimization Package

Several computational tools interconnect in the optimization procedure are listed below:

- UPS3D: 3-D parabolized Navier-Stokes code; inviscid calculation only (Ref. 1)
- NPSOL: numerical optimization code6; a sequential quadratic programming algorithm in which the search direction is the solution of a quadratic programming subproblem
- HYPGEN: hyperbolic grid generator7; a sufficiently fast and robust to operate within an automated optimization environment.
- LHF: sonic boom extrapolation code (Appendix A); a routine based on Whitham's F-function and the equal-area rule8
- SAMGRID: wing/body surface grid generator (Appendix B); a sufficiently fast and robust to operate within an automated optimization environment
- DB: sonic boom loudness calculation; a code gives perceived loudness (PLdB) of the sonic boom can be determined by Stevens’ Mark VII method9 which involves Fast Fourier Transform on the energy spectrum of the sonic boom

This CFD optimization package is robust and efficient on Cray-YMP. The application of this package will be described in the following sections.

2.2.2 Aerodynamic and Sonic Boom Optimization

The optimization design package was exercised using a recently-developed low-boom wing-body configuration, Boeing 1080-991 (also called Haglund model), designed by George Haglund. This optimization technique was applied separately to the two objectives of high aerodynamic performance and low sonic-boom loudness.

For aerodynamic enhancement, control points are set on the cambers of the wing, with the thickness kept fixed. The left figure below shows the differences on a inboard airfoil section of the original and the modified. The polar plot shows the improvement of L/D of the modified configuration over the original by 3.8%. The right figure below shows that the modified wing had less wave drag than the original one at the leading edge. This means
that the leading thrust is improved by the optimization process. The whole process takes about 4 CPU hours on Cray-YMP.

For sonic boom improvement, F-function was employed as an entity to define the equivalent area distribution and sonic boom shape. The original Haglund model was supposed to give a flat-top pressure waveform at the ground. However, calculations showed that the waveform had an intermediate shock followed right after the bow shock; whereas the flat-top waveform would have no intermediate shock. The design code redistributed the equivalent area of the fuselage (without changing the wings), and re-captured the flat-top characteristic of the pressure waveform. The figure below compares the sonic boom signatures among the original, optimized, and target flat-top. Due to the sensitive nature of the configuration, the change of the configuration will not be shown here. The details of this optimization methodology and results were considered as sensitive materials and were presented in the 2nd Annual Sonic Boom Workshop.

2.2.3 Drag Minimization on Haack-Adams Body

The purpose of this study was threefold:

• to search for a design method to minimize the drag of a supersonic projectile
• to demonstrate the capability of the CFD optimization package described above
• to search for computational grid density effect on optimization performance

The baseline configuration chosen for this study was called Haack-Adams body, a body of revolution with a pointed nose and a base of finite area. This body was thought to be the minimum-drag body under the slender body theory. Wind-tunnel data were available for CFD validation. The method of optimization made use of the Fourier Sine expansion, which had three main advantages over the traditional techniques based on shape functions and control points:

• The volume of the body was fixed without putting external constraints. External constraints cost more computational time. For some cases, fixed volume is not feasible.
• Global minimum was search.
• Number of design variables was substantially reduced.
The figure below summaries the result of this study. The nose of the body was trimmed to reduce the wave drag. Since the total volume was constrained, volume was added near the end of body. Total wave drag reduction was by 6%. The results were presented in a AIAA meeting\textsuperscript{12} and published in Journal of Aircraft Vol. 32, No. 1, Jan/Feb. 1995.

\subsection*{2.3 Low-Boom Wind-Tunnel Configuration (Ames Model 3)}
Efforts were made to design a new wing/body/nacelle configuration, which had a lower sonic boom relative to the baseline, 1080-911 from Boeing Company, of low boom HSCT concept. The CFD optimization package described in Section 2.2.1 were employed to modify this baseline configuration. The result of the optimization was used to build a wind-tunnel model, Ames Model 3, tested at Ames 9'x7' wind tunnel in June 1993. Due to the sensitive nature of the configuration, no planform shapes will be shown here. However, the left and right figures below show the computational grid and the optimization result, respectively. The plot at the lower right-hand corner of the right figure shows the sonic booms of the baseline and Model 3 respectively. The baseline configuration has a loudness level about 100 PLdB; whereas Model 3 has about 92 PLdB. The results of this research were presented in the 3rd Annual Sonic Boom Workshop\textsuperscript{13}.
2.4 Oblique-All Wing (OAW) Computation and Design

Oblique flying-wing\textsuperscript{14} is an alternative supersonic aircraft concept. Ames, Boeing, Douglas, and Stanford University joined and formed a design team in 1992 to investigate the feasibility of OAW for commercial use. The study included aerodynamic performance, stability, structure, landing gear, airplane exits, and airport regulations. The design team decided to build a wind-tunnel model for wind-tunnel testing in June 1994. My job was to provide Navier-Stokes CFD supports and, if possible, optimization results. The figure below shows some of the wings that were analyzed since the beginning of this study.

The flow solver being used was Overflow code, a 3-D Navier-Stokes code using the diagonal with ARC3D algorithm\textsuperscript{15}. One of the most challenging works of this project was to reduce the separation on the left wing (trailing wing). The separation on the upper surface of the wing and the corresponding vortices are shown in the left side of the figure below. It was found that bending of the wing could abate the separation, as well as improve the lift-to-drag ratio. The right side of the figure shows a weaker separation pattern on the ended
Due to the sensitive nature of this study, the results can only be presented in the weekly group meetings at Ames and a controlled distributed NASA Contractor Report.

3 CURRENT WORK/RESULTS

Currently, research effort was concentrated on one theme that is sharpening the tools for HSCT design. Three research topics are focused: near-field CFD calculation and sonic boom softening of Boeing Reference-H, improvement of sonic boom extrapolation, and aerodynamic design on parallel computer.

In order to study and design a real complex aircraft, a relatively fast CFD technique has to be developed for optimization environment. Coupling a fast space-marching code and a time iterative code with overset gird concept can take the advantage of marching code at the fuselage/wing region and solve the complex flow field near the wing/nacelle region at the same time.

A very efficient wave propagation code for mid-field sonic boom prediction has been developed based on the method of characteristics. This code solves the Euler equations for 1.2 minutes on Cray-YMP; whereas, the axisymmetric CFD method described in Section 2.1.1 takes 40 minutes on the same computer.

Number crunching problems, like CFD calculations, on parallel machines can be efficiently done in today’s computing environment. This may lead to the future of aerodynamic research and design. In order to exercise HSCT design on parallel computers, a nonlinear optimization routine has been developed for a network based parallel computer system in which a cluster of engineering workstations serves as a virtual parallel machine.

3.1 Sonic Boom and Performance Study of Reference-H

Research effort on low-boom configuration concept has been invested for the past four years. A new proposed route structure for HSCT’s incorporating supersonic corridors over land and water has relaxed the sonic boom constraint somewhat. The objective of this study is twofold. First is to exercise the methodology of combining two different CFD codes to solve the near-field solution of a realistic HSCT configuration in an efficient and accurate manner. Second is to reduce the sonic boom loudness of a performance configuration concept, Reference-H, without jeopardizing the aerodynamic performance. The basic components of Reference-H are a fuselage, a pair of swept wings, and four nacelles.

3.1.1 Reference-H Near-Field Study

The CFD codes used in this study are the UPS3D code and the OVERFLOW code. Both CFD codes has been described in Section 2.1.1 and 2.4, respectively. The former is an efficient space-marching code. However, it fails in the region where subsonic pocket exists; especially in the region of the wing/nacelle integration. The latter is a time-iterative code with Chimera overset grid concept, which makes the code more viable in solving the
region of wing/nacelle integration. In this study, only inviscid flow is considered. Figure below summarizes the result of the CFD calculations.

![Reference-H UPS/OVERFLOW Interface](image)

The near-field solution is studied for the case of Mach number 2.4 and angle of attack 4.5 degrees. Wind-tunnel data of the Reference-H validate the CFD method. Study shows that flow particles turn significantly over the outer nacelle compared with the inner nacelle. It indicates that the effect of the nacelle orientation might improve the aerodynamic performance.

### 3.1.2 Sonic Boom Softening

The sonic boom of the Reference-H configuration is also obtained. The calculation shows that the boom is an N-wave of 104 PLdB with 2.5 psf. bow shock on the ground. Details of the sonic boom prediction technique can be found in Ref. 10. Boom modification for performance aircraft is very much different from the low-boom aircraft for cruise Mach number and lift are higher. Therefore, the technique developed previously cannot be strictly applied to Reference-H. However, changing the equivalent area can be helpful. The result of this study was presented in the 4th Sonic Boom Workshop. Another approach to reduce the boom is by experimenting the sweep angle. The figure above show one of the exercises done on the Ref-H. This exercise successfully shows Boeing how much boom
reduction can be achieved by redistributing the lift. An closer on-going technology communication with airframe industry is needed in order to achieve the goal of sonic boom softening on performance aircraft. A team consisting myself and other personnels from Boeing and NASA Langley has been formed to achieve the goal.

### 3.2 Sonic Boom Mid-Field Extrapolation (WPSYM)

In the beginning of 90's, sonic boom extrapolation technique was still relied on the linear theory developed in the 60's for the nonlinear techniques were computationally expensive. Today, a fast and accurate sonic boom extrapolation methodology is needed to bring the sonic boom extrapolation technique up to the 90's standard for HSCT design. The objective of this study is to develop an efficient and accurate higher-order computational method, solving the Euler equations, for supersonic aero-acoustic wave propagation.

An axisymmetric wave propagation code (WPSYM) has been developed for mid-field sonic boom extrapolation. This propagation code has been demonstrated as an efficient and accurate tool over the previous CFD method, described in Section 2.1.1 and Ref. 4, on a generic wing-body configuration. The figure below shows that a 3-D near-field solution

![Sonic Boom Propagation](image)

is obtained from UPS3D code; the result is then interfaced to two axisymmetric sonic boom extrapolation codes, namely, the axisymmetric version of UPS3D and the recent wave propagation code (WPSYM). The former takes 40 minutes on Cray-YMP, and the latter takes 1.2 minutes on the same machine. The x-y plot in the figure compares the numerical extrapolation results to wind-tunnel data. The result has been shown in NASA Technical Highlight and the methodology has been presented in the 4th Annual Sonic Boom Workshop at NASA Langley in June 1994.\(^\text{16}\)
3.3 Optimizer on PVM (IIOWA)

Moving to the world of parallel computing, the aerospace industry needs a numeric optimization tool in the parallel environment. One of the promising parallel computing concept is the network-based distributed computing. The Parallel Virtual Machine (PVM) is a software package that allows a heterogeneous network of parallel and serial computers to appear as a single concurrent computational resource. PVM allows users to link up engineering workstations to work as a single distributed-memory (parallel) machine. Merritt Smith and I wrote a manual on PVM for beginning users. A copy of the manual is attached in Appendix C.

A parallel optimizer based on nonlinear Quasi-Newton method has been developed and coupled with an efficient CFD code for basic aerodynamic design and study. This optimizer is called IIOWA (parallel Optimizer With Aerodynamics). The figure below is a demonstration of IIOWA. A Boeing arrow wing/body configuration is chosen in this study. The fuselage radius is changed so that the wave drag is minimized. The parallel CFD optimization process takes 24 wall-clock hours on 4 SGI workstations to reduce the wave drag by 6.5%. The optimized result is a “coke bottle” shape fuselage, as expected by supersonic area rule. The convergence history of the optimization process is also shown in the figure. The optimizer is also coupled with a parallel CFD code, MEDUSA, to perform viscous 2-D multizone airfoil optimization supported by overset grid concept. The results will be presented at NASA CAS conference in March 1995.

3.4 Oblique All-Wing (OAW): CFD support

The OAW design team has asked for CFD support on the latest configuration OAW-3 from which a wind-tunnel model has been built and tested at Ames in June 1994. The figure below shows the chimera grid topology on the OAW-3 with fin. The design team want to compare the CFD result with the result from pressure sensitive paint (PSP). Therefore,
CFD calculations have to be done prior to the wind-tunnel test because color map from CFD result is need for PSP calibration.

4 SUMMARY

The computational tools for sonic boom prediction, aerodynamic calculation, and configuration design of the current HSCT concept have been validated and applied to build wind-tunnel model for further testing and validation. The techniques developed in this five-year research and their applications, such as sonic boom prediction technique (Section 2.1), design of Ames Model 3 (Section 2.3) by CFD optimization (Section 2.2), and sonic boom softening for performance configuration (Section 3.1), have clearly shown support to the HSRP as it moved to its phase two period.

An accurate sonic boom extrapolation tool has always been an issue. It is because the flow phenomena in the atmosphere are nonlinear, but the common technique for extrapolation is linear acoustic theory developed in the 60's. On the other hand, CFD technique is too computationally expensive. Recently, a fast and accurate sonic boom extrapolation methodology (Section 3.2), solving the Euler equations for axisymmetric flow, has brought the sonic boom extrapolation technique up to the 90's standard.

Parallel computing is a fast growing subject in the field of computer science because of the promising speed in number crunching computations. A new optimizer (Section 3.3) for parallel computing concept has been developed and tested for aerodynamic drag minimization. This optimimer is also coupled with a parallel CFD code so the whole optimization process is parallel. This is a promising method for CFD optimization making use of the computational resources of workstations, which unlike supercomputers spend most of their time idle.

Finally, the OAW concept is so attractive because of its overall performance in theory. In order to fully understand the concept, a wind-tunnel model is built. CFD Navier-Stokes calculations helps to identify the problem of the flow separation (Section 2.4), and also help to design the wing deflection for roll trim and alleviating the flow separation.
5 References


Appendix A

LHF (Fortran Listing)
PROGRAM LHF

This program calculates
1) the Lightbody F-function on body surface,
   - input data
2) the pressure signature at given distance R1
3) the loadless level of the sonic boom at R1

INPUT
- LHFILE (input parameter
area.in(3) Equivalent area distribution, (INAREA=0)
(f.dat) F-Function distribution, (INAREA=2)
area.of(3) Equivalent area distribution, (INAREA=0)
fa1.dat (F-Function due to lift, (LIFT=1)
area.out(3) Equivalent area distribution and its derivative,
   if out(3) F-Functions on the body surface and at distance R1
   if out(3) Pressure Signature at distance R1
   Icurve_F(14) Integral curve of the shifted F-function.

DATA

PARAMETER (NMAX=220, NLMAX=1, NMAX=351, NMAX=900)
DIMENSION (DOX, DOY, NMAX), (DOX, DOY, NMAX), (DOX, DOY, NMAX)
REAL (NMAX), (NMAX), SP(NMAX), TAB(NMAX), PTAB(NMAX)
COMMON/PAR, PRMF, PFAC
LOGICAL BWDF
BWDF = .FALSE.

OPENUNIT. FILE='LHF.in', STATUS='OLD'

Read the input parameters
PANEL : PAR, PRMF, PFAC, RO, R1, INAREA, LIFT, TOWN
READ(1,PARA)
WRITE(6,PARA)

Input free-stream Mach number = PRMF
If TOWN = 0, sonic boom varies time, else varies distance
If the surface grid contains the whole configuration, or
only half-planes or only quarter-plane?
PFAC = 1, 1 Whole plane
PFAC = 2, 1 Half-plane
PFAC = 4, 1 Quarter-plane
R1 will be the distance where the signature is captured.
If read in area distribution, INAREA = 0
read the grid
read the wing-body case
read F-function
read signature at R0
IU = 4, ATAN(1.)
JLM = NMAX
ZD = 361
If R0 = 0, we read the pressure signature at R0 and extrapolate
IU = 400.0, 0.0, GOTO 790
Find the area distribution of body configuration (sample case).
IF(INAREA LT 0) THEN
CALL WIND(JLM, S, TAU)
WHDF = .TRUE.
CALL CORE(JLM, S, TAU)
CALL WIND(JLM, S, TAU)
CALL SHEAR(JLM, S, TAU)
CALL BULEY(JLM, S, TAU)
GOTO 270
ENDIF

Read in the given area distribution
IF(INAREA EQ 0) THEN
OPEN(UNIT=2, FILE='area.in')
DO 50 J=1,NMAX
READ(2, END=75) TAU(J), S(J)
50 CONTINUE
75 CONTINUE
CLOSE(2)
DO 90 J=1,JDM
OPEN(UNIT=3, FILE='area.in')
READ(3, END=110) TAU(J), S(J)
90 CONTINUE
GOTO 270
ENDIF

Read in the F-function or define a F-function by calling FUNC
and integrate out the equivalent area by calling INAREA
IF(INAREA EQ 2) THEN
CALL FUNC(TAU, PTAB, JDM)
OPEN(UNIT=4, FILE='f.dat')
DO 100 J=1,NMAX
READ(4, END=110) TAU(J), PTAB(J)
100 CONTINUE
ENDIF
CALL DISTARC(TAU, FTau, J-1, TAU, FTau, JDIM, 10., 0)
CALL DAREA(S, FTau, TAU, JDIM)
GOTO 270
ENDIF
DO 200 J-1, JDIM
   S(J) = FFAC*8(J)
   TAU(J) = G(J, J)
220 CONTINUE
270 CONTINUE

CALL EAREA(S, FTau, TAU, JDIM)
GOTO 270
ENDIF
DO 220 J-1, JDIM

S(J) = PFAC*S(J)
TAU(J) = T(J)
220 CONTINUE
270 CONTINUE

IF(LIFT DO 1) CALL BPWDFC(JDIM, S, TAU)

DO 100 J-1, JDIM

A(J) = S(J)/PI
340 CONTINUE
100 CONTINUE

CLOSE(13)
STOP
END
SUBROUTINE LIGHT1(TAU, R, SP, N, F Tau, FTau)

PI = 4. * ATAN(1.)
BETA = 5ORT(FHAC**2-1.0)
TAU(1)=0.
FTAU(1)=0.

DO 95 N=1,N

FTAU(N)=0.

DO 100 J=1,N

IF(ABS(R(J)), LE.1.E-10) THEN
  Z1 = 1.E+16
  F4 = 0.
  GO TO 95
ENDIF

ABS = 2.0/BETA*R(J)
AB1 = ABS(AB1)
F1 = SQRT(AB1)
F2 = SP(1)-SP(1-1)
F3 = F1+F2
F4 = F3/(2.0+F1)
F5 = (TAU(J)-TAU(J))/BETA*R(J)
FLD = 1.0
IF (Z1.LT.XI) GO TO 94
IF (FLD.LT.4.0) GO TO 97
IF (FLD.GE.4.0) GO TO 98

95 IIZI = 0.

IF (FLD.LT.2.0) THEN
  ZF = (Z1.LT.4.0) GO TO 97
  FTAU(J) = FTAU(J)+ZI*F4
  GO TO 99
ENDIF

96 ZI1 = 0.

IF (FLD.JE.4.0) THEN
  FTAU(J) = FTAU(J)+ZI*F4
  GO TO 99
ENDIF

97 BZ1 = 0.2937*XI+XI*.2175*XI+.7531

98 FTAU(J) = FTAU(J)+BZ1*F4

99 CONTINUE

100 CONTINUE
RETURN
END
SUBROUTINE EQUAREA(EDI1,LDIM,JDIM,X,Y,Z,EMAX,LMAX,LMAX,LMAX)

This subroutine finds the cross-section area of a surface grid which has symmetry plane at Y-axis. For each marching station (for each x) the area is found. The area is approximated by trapezoidal rule.

DIMENSION X(EMAX,LMAX,LMAX),Y(EMAX,LMAX,LMAX),Z(EMAX,LMAX,LMAX)

REAL S(3,3)

DO 10 J=1,LDIM
   SAREA = 0.
   DO 5 K=2,LDIM
      X = X(1,LDIM,J)-X(K-1,LDIM,J)
      ADD = X(1,LDIM,J)+X(K-1,LDIM,J)
      SAREA = SAREA + 0.5*X+Y
   CONTINUE

S

The unwanted base area.

X = X(1,LDIM,J)-X(1,LDIM,J)
ADD = X(1,LDIM,J)+X(1,LDIM,J)
BASE = ABS(0.5*ADD)+B

The area surrounded by half of the plane

S(J) = ABS(SAREA)-BASE

10 CONTINUE

RETURN

END
SUBROUTINE FFN(I, P Mach, NP, RO, RI, TORX)

This program uses F-function theory to predict the pressure
signature at far field when an initial pressure signature is given

PARAMETER (NMAX=1400)
DIMENSION P(NP), X(NP), P(NMAX), P(NMAX)
DIMENSION Y(NMAX)
DIMENSION Dfv(3)

OPEN (UNIT=14, FILE='p.out')

Input of initial parameters and pressure signal

IF (RO > 0) THEN
  OPEN (UNIT=4, FILE='p.RO')
  DO 15 I=1,NMAX
    H(i) = H(i) + 10*Y(i)
    CONTINUE
  15 CONTINUE
END IF

Define the parameters used in the F-function theory

GAMMA = 1.4
B = SQRT(PMACH**2 - 1)
CAP = GAMMA + PMACH**2/(SQRT(2)*B**1.5)

IF (B > 0) THEN
  OPEN (UNIT=8, FILE=ROX)
  DO 30 I=1,RO
    F(I) = SQRT(SQRT(2*B*B)*P/I)/(GAMMA + PMACH + PMACH - PMACH)
    X(I) = X(I) - B*R + CAP*SSR + P/I
    CONTINUE
  30 CONTINUE
END IF

Y is transposed coordinates

WRITE(13,48)
DO 49 I=1,NP
  X(I) = X(I) - CAP*SSR + P/I
  WRITE(13,49) X(I),P(I)
  49 CONTINUE

Find the largest and smallest values of T

TMAX = -1.84
TMEN = 1.84
DO 55 I=1,NP
  IF (X(I) < TMAX) TMAX = X(I)
  IF (X(I) < TMEN) TMEN = X(I)
  CONTINUE

Print out the integral curve of the shifted F-function

CALL INTR(NP,T,F)

Need to march in Y-direction, define the step

TSTEP(1) = TMIN
TSTEP(2) = TMAX - TMIN
TSTEP(3) = TSTEP(2) + 1 + TSTEP(1)
CONTINUE

March through the shifted F-function, check area balance and
place the shock

CALL MARCH(NMAX,NP,T,Y,TSTEP,F)

Obtain the solution

NOTE: FOR TOR > 0, the sonic boom is in the form (P-Plat) vs time
as shown in figure 1. For the form (P-Plat)/Plat vs distance.

DO 150 I=1,NNP
  P(I) = GAMMA + PMACH + PMACH + P(I)/SQRT(2*B*B)
  Y(I) = Y(I) + B*R
  CONTINUE

Make the data points evenly distributed manner and
scale the sonic if desired

DO 180 J=1,NNP
  X(J) = X(J)*PSCALE
  P(J) = P(J)*PSCALE
  CONTINUE

Atmospheric aspect

ALT = Altitude
A0 = speed of sound at ground, in ft/sec
A0 = reference pressure ln/ft^2 = SQRT(P0*Ps)
Ps = pressure at the ground
Ps = pressure at flight altitude
P0 = SQRT(P0)
VEL = FMA/2
TRE = X(1)/VEL
IF (NXC < 5), THEN
  DO 260 I=1,NNP
    X(I) = X(I)/VEL - TRE
    P(I) = P(I)*P0
    CONTINUE

The signal (DP vs Time) is calculated, use a empirical program to
calculate the rise time, and embed the rise time into the signature.
Note: Unit used is still the stupid English unit!
CALL WISELINE(FWAVE, P, X, NP, ALT, TRISE)

Obtain the noise level
CALL NOISE(DBLVL, X, P, NP)

Write the dB(PL) value out
WRITE(14,500)DBLVL(1),DBLVL(2),DBLVL(3)

500 FORMAT
   '('Noise level ',F10.4,'/dB',5X,F10.4,'/dB(A)',3X,F10.4,'/dB(C)')

Write the sonic boom
WRITE(14,555)R1

555 FORMAT(28DF(10,1,8,12),2X,F10.4)  
DO 670 I=1,NP
   WRITE(14,700)X(I),P(I)
670 CONTINUE

700 FORMAT(3X,E20.8,2X,E16.6)

CLOSE(14)

RETURN
SUBROUTINE INTF(NP,Y,F)

C This program print out the integral curve of the shifted F-function
D DIMENSION F(NP),Y(NP)
C OPEN(UNIT=34,FILE=\'icurve_f\',FORM=\'FORMATTED\')

C SUMF = 0
C DO 100 J=1,NP
  CV = T(J)-T(J-1)
  SUMF = SUMF + 0.5*DY*(F(J)+F(J-1))
C WRITE(34,130)Y(J),SUMF
C 100 CONTINUE
C FORMAT(42R# Integral curve of the shifted F-function)
C C COSE (34)
C RETURN
END
SUBROUTINE SHEET(NMAX,NP,Y,TSTP,F,INDEX,FS,YS,INST)

DIMENSION Y(NP),T(NP)
DIMENSION TSTP(NMAX)
DIMENSION INDEX(NMAX),FS(40)
COMMON/ZSHOCK/,INST

YEND = Y(NP)
FIRST = 1
DO 500 J = 1,NMAX
YS = YSTP(J)

C Get the points on the curve for integration, start searching from
C TS to TS + TSTP(J)
C
CALL POINT(NP,Y,INDEX,FS,YS,INST)

C After obtaining the integration points, we can integrate and
C find the area
C
IF(INSCT.GT.2) THEN

IF(INST.EQ.3) INST = 3
IE = INDEX(INST)
CALL AREA(NP,Y,F,TS,FS,IS,IE,IFLAT1)
ELSE

The tail shock is already formed, leave program
IF(INST.EQ.1 .AND. YS.GT.YEND-1.65) RETURN
FIRST = 1
GOTO 500
ENDIF

IF(FIRST.GT.0.) IFLAT1 = IFLAT2
IF(IFLAT2.EQ.0.) RETURN
FIRST = -1.
ENDIF

IF(IFLAT = 0, IS is the point that have area balanced.
C IF IFLAT2 and IFLAT are in different sign, i.e.,
C the common point should be between 1 and 1-1.
C Use bisection method to find the correct point Y(ISTART)
C
IF(IFLAT1+IFLAT2 .LT. 0.) THEN

Y1 = YSTP(J-1)
Y2 = YSTP(J)
NC = 500
DO 300 I = 1,NC
YS = 0.5*(Y2-Y1)
CALL POINT(NP,Y,F,INDEX,FS,YS,INST)
IF(INST.EQ.3) INST = 3
IE = INDEX(INST)
CALL AREA(NP,Y,F,TS,FS,IS,IE,IFLAT)
IF(IFLAT.EQ.0) RETURN
IF(IFLAT.GT.0 .AND. YS.GT.YEND) RETURN
ELSE
IF(IFLAT1 = IFLAT2
ELSE
IF(IFLAT1 = IFLAT2
ENDIF

200 CONTINUE
WRITE(*,*) 'After ',NC,' steps of bisection'
RETURN
ELSE
IFLAT1 = IFLAT2
GOTO 500
ENDIF
C
C FIRST = 1
500 CONTINUE
RETURN
END
SUBROUTINE POINT(NP, Y, F, INDEX, FS, YS, INSC)
DIMENSION Y(NP), F(NP)
DIMENSION INDEX(40), FS(40)

C Find the points FS on the F-function when YS is given.
INDEX = the index runs from 1 to NP
INSC = # of points being intersect, at least 1 point to do integration

IF (YS .GT. Y(1)) THEN
   INSC = INSC + 1
   INDEX(INSC) = NP
ELSE
   PRINT *, 'ZEROOOOOOO,YS,Y(1),I'
   PRINT *, 'ZEROOOOOOO', YS, Y(1), I
   INSC = 0
END IF

DO 100 I = 2, NP
   FAC3 = YS - Y(I)
   IF (FAC3 .LE. 0.0) THEN
      IF (ABS(Y(I) - Y(I-1)) .LT. 1.E-14) THEN
         WRITE(*,'(I4,4X,3F12.6)') Y(I), Y(I-1), Y(I-2)
      ELSE
         WRITE(*,'(4X,S76)') 'ZEROOOOOOO', YS, Y(I), I
      END IF
      INSC = 0
      RETURN
   END IF
   INSC = INSC + 1
   SL = (F(I-1) - F(I))/((Y(I) - Y(I-1))
   FS(INSC) = F(I-1) + SL*(YS - Y(I-1))
   INDEX(INSC) = I
100 CONTINUE

IF (YS .GT. Y(NP)) THEN
   INSC = INSC + 1
   FS(INSC) = F(NP)
   Y(INSC) = YS
   INDEX(INSC) = NP
ELSE
   PRINT *, 'ZEROOOOOOO', YS, Y(I), I
END IF

RETURN
END
SUBROUTINE AREA(HF,Y,F,YS,FS,IS,IE,IFLAT)

DIMENSION Y(HF),F(HF),FS(45)
COMMON/SHOCK/ INSGT

C Fine the integral of F by trapezoidal rule
C Integrating from Y(IS) to YS. E1 is area that from YS to Y(IS)
C and E2 is the area that from Y(IS) to YS. Thus E2 should be
C subtracted out and E1 should be added in

560 E1 = 0.5*(Y(IS)-YS)*(F(IS)+F(1))
561 E2 = 0.5*(YS-Y(IS))*(F(IS)+F(IS+1))
562 AREAL = E1
563 IE = IE-1
564 DO 10 I = 15,IE
565 SLAP = 0.5*(Y(I-1)-Y(I))*(F(I-1)+F(I))
566 AREAL = AREAL + SLAP
10 CONTINUE

C A > AREAL + E2
568 A = AREAL + E2

IF(A.GT.0) IFLAT=1
IF(A.LT.0) IFLAT=-1
IF(MAX(A).LT.0.0) IFLAT=0

574 RETURN
575 END
SUBROUTINE WBODY(JDIM,S,TAU)
C This subroutine finds the area distribution of
C the wing-body configuration.
DIMENSION S(JDIM),TAU(JDIM)
C
C
PI=4.*ATAN(1.)
ANG=21.*PI/180.
ANG1=80.*PI/180.
DI=25.52/PI/180(JDIM-1)
C
S(1) = 0.
DO 2 J=2,JDIM

TAU(J) = TAU(J-1)+DX

TTT = TAU(J)-7.01
IF(TTT.GT.0.) TTT'=0.
RR=0.54-0.011*TTT**2
S(J) = PI*RR*RR

IF(TAU(J).GT.8.31 .AND. TAU(J).LT.12.25) THEN
AA = 4.*0.5*0.05*TAN(ANG1)*(TAU(J)-8.21)**2
S(J) = S(J) + AA
ENDIF

IF(TAU(J).GT.12.25 .AND. TAU(J).LT.15.77688849) THEN
B1 = 0.05*(14.29-TAU(J))
B2 = 2.91*(TAU(J)-12.25)/(15.77688849-12.25)
B3 = (TAU(J)-8.21)*TAN(ANG1)-B2
H1 = 0.05*B1/TAN(ANG1)
AA = 4.*0.5*H1*H1+0.5*(H1+B3)**2
S(J) = S(J) + AA
ENDIF

IF(TAU(J).GT.15.77688849 .AND. TAU(J).LT.16.29) THEN
AA = 4.*0.5*0.05*TAN(ANG1)*(16.29-TAU(J))**2
S(J) = S(J) + AA
ENDIF

IF(TAU(J).GT.16.29 .AND. TAU(J).LT.17.52) THEN
SLOP=(0.15-0.54)/(17.52-17.29)
RBR=0.54+SLOP*(TAU(J)-17.29)
S(J) = PI*RBR**2
ENDIF

IF(TAU(J).GT.17.52 .AND. TAU(J).LT.17.93) THEN
S(J) = PI*0.15*0.15
C
CONTINUE
RETURN
END
SUBROUTINE CONE(JDIM,5,TAG)
C     This subroutine finds the area distribution of the cone-cylinder
C     with half-angle 3.24 degree and 8.8 units of length.
DIMENSION S(JDIM),TAG(JDIM)
PI=4.*ATAN(1.)
ANG = 3.24*PI/180.
DX = 16./FLOAT(JDIM-1)
S(1) = 0.
TAU(1) = 0.
DO 2 J=2,JDIM
   TAU(J) = TAU(J-1)+DX
   IF(TAU(J) .GE. R) THEN
      R = TAU(J)*TAN(ANG)
   ELSE
      R = 8.6*TAN(ANG)
   ENDIF
   S(J) = PI*R*R
2  CONTINUE
RETURN
END
SUBROUTINE SEAMS(JDIM, S, TAU)

C This subroutine finds the area distribution of the Sears-Haskin body

DIMENSION S(JDIM), TAU(JDIM)

C # of point on body + # of point on sting = JDIM

JBDY = JDIM(2,1)

JHING = JDIM - JBDY

PI = 4. * ATAN(1.)

BL = 1.

TOTA = 1.9 * BL

F = 21.5

DTHETA = PI/FLOAT(JBDY-1)

DS = BL/FLOAT(JBDY-1)

RMAX = BL/(2.*F)

S(1) = 0.

TAU(1) = 0.

C Consists of Sears-Adams body

WRITE(6,'(*)') 'Input Amax/Amax'

READ(6,*) AA

WRITE(6,'(*)') 'Input Rmax'

READ(6,*) AA

RMAX = AA*BL

CONST = AA/PI

C1 = 1./2.*(2.*RMAX/BL - 1.)

DO 3 J=2,JBDY

C THETA = PI-DTHETA*FLOAT(J-1)

TAN(J) = (1.435*(THETA)**2)/2.

TAU(J) = FLOAT(J-1)*DS

THETA = ATAN2(2.*TAU(J)/BL - 1.)

C Sears-Haskin body

POS = SIN(THETA)***(1.

P = RMAS*SORT(ABS(POS))

S(J) = P1+R

3 CONTINUE

C Add a sting

DS = (TOTAL-BL)/FLOAT(JHING)

DO 5 J=JBDY+1,JDIM

TAU(J) = TAU(J-1) + DS

S(J) = S(J-1)

5 CONTINUE

RETURN
END
SUBROUTINE BULLET(JDIM,S,TAU)

This subroutine finds the area distribution of a bullet with a form

\[ R = 0.4 + \tan(1) \]

\[ BL = 4 \]

\[ GAMA = 0.65 \]

\[ RBASE = 0.25 \]

\[ A = RBASE/(1+GAMA) \]

\[ TOTLEN = BL + 2 \times BL \]

\[ DX = TOTLEN/FLOAT(JDIM-1) \]

\[ S(1) = 0 \]

\[ TAU(1) = 0 \]

DO J=2,JDIM

\[ TAU(J) = TAU(J-1) + DX \]

IF(TAU(J) \geq BL) THEN

\[ R = RBASE \]

ELSE

\[ R = A \times TAU(J)^2 \]

END IF

END

CONTINUE

RETURN

END
SUBROUTINE WING(JDIN, S, TAU)

C This subroutine finds the area distribution of the low-aspect-ratio wing

DIMENSION S(JDIM), TAU(JDIM)

PI = 4.0 * ATN(1.0)
STING = 0.
DX = 1.0 / (FDIM(JDIM - 1))
S(1) = 0.
TAU(1) = 0.
DO 2 J = 2, JDIM
1 TAU(3 - J) = TAU(J - 1) + DX
2 IF (TAU(J) .LT. 0.5) THEN
3 Z = (PI / 12.5) * (TAU(J) - 0.5 * TAU(J) * TAU(J))
5 S(J) = Z
6 IF (TAU(J) .GT. 1.0 .AND. JD(J) .LT. 7.0897) THEN
8 STING = PI * 0.0625 * 0.0625
10 S(J) = STING
12 ENDIF
14 ELSE
15 STING = PI * 0.0625 * 0.0625
17 S(J) = STING
19 CONTINUE
21 RETURN
22 END
SUBROUTINE BFUNC(JDIM,TAU)

This subroutine obtains the B-function from fort.10 and add it

DIMENSION (JDIM),TAU(JDIM),B(NMAX),Z(NMAX)

COMMON/PAR/ BMAX,PFAC

OPEN(UNIT=33,FILE='coeff.dat',FORM='FORMATTED')

READ(33,12) READ(33,12) READ(33,12) READ(33,12)

READ(33,15,END=17) E(1),CL,CD,SLO,SI(1),CH

READ(33,15,END=17) E(1),SI(1)

B(I) = B(I)*PFAC

10 CONTINUE

11 CONTINUE

12 CONTINUE

13 FORMAT(6_13.5)

14 CONTINUE

CLOSE(33)

NPOINT = 1-1

OPEN(UNIT=13,FILE='bga.dat')

DO 30 I=1,NPOINT

WRITE(33,1*) A(I),B(I)

20 CONTINUE

ISTART=1

DO 50 J=1,JDIM

DO 30 I=1,NPOINT

IF(ABS(X(I))>TAU(J)) .LE.1.0E-10 THEN

S(1)=S(1)+B(I)

ISTART=1

GOTO 40

ENDIF

IF(X(I) GT TAU(J)) THEN

IF(I.EQ.1) THEN

BP=0.

IF=0.

ELSE

IF=I-1

ENDIF

SLOPE=(B(I)-BF)/(X(I)-XF)

BF = B(I) + SLOPE*(TAU(J)-X(I))

S(2) = S(2) + BF

ISTART=1

IF(I.EQ.1) ISTART=1

GOTO 40

ELSE

IF(I.EQ.NPOINT) GOTO 30

S(1) = S(1) + B(NPOINT)

ISTART=1

GOTO 40

ENDIF

30 CONTINUE

40 CONTINUE

50 CONTINUE

RETURN

END
SUBROUTINE KB(JDIM, SP, TAU)
C This subroutine obtains the wing-body interference correction
C and add it into the derivative of equivalent area
C This is a test for wing-body case
DIMENSION SP(JDIM), TAU(JDIM)
COMMON/PAR/ FMAC, PFAC
C
DO 10 J=1,JDIM
IF(TAU(J).GE.8.2L .AND. TAU(J).LE.12.25) SP(J)-SP(J)-4.*.05*.54
IF(TAU(J).GT.12.25 .AND. TAU(J).LE.16.29) SP(J)-SP(J)-4.*.05*.54
10 CONTINUE
RETURN
END
SUBROUTINE FUNC(TAU,FTAU,JDIM)

DIMENSION TAU(JDIM),FTAU(JDIM)

NAMELIST /FUNC/ YF,ELAM,C,B,D,E,BL,TR,DEL

C.Read the input parameters
READ(3,FFUNC)
WRITE(6,FFUNC)

TAU(1)=0.
FTAU(1)=0.
FTY/FTY/FLOAT(JDIM-1)
DO 10 J=2,JDIM

TAU(J)=TAU(J-1)+DY
IF(TAU.EQ.0.) GOTO 6
IF(TAU(J).LE.YF/2.) FTAU(J)-2.*TAU(J)/YF
IF(TAU(J).GT.YF/2.0.AND.TAU(J).LE.YF)
   & TAU(J)=C*(TAU(J)/YF-1.)-B*(TAU(J)/YF-2.)

IF(TAU(J).GT.YF/2.0.AND.TAU(J).LE.YF)
   & TAU(J)=C*(TAU(J)/YF-1.)-B*(TAU(J)/YF-2.)

IF(TAU(J).LE.YF/2.) FTAU(J)-2.*TAU(J)/YF
IF(TAU(J).GT.YF/2.0.AND.TAU(J).LE.YF)
   & TAU(J)=C*(TAU(J)/YF-1.)-B*(TAU(J)/YF-2.)

IF(TAU(J).LE.YF/2.) FTAU(J)-2.*TAU(J)/YF
IF(TAU(J).GT.YF/2.0.AND.TAU(J).LE.YF)
   & TAU(J)=C*(TAU(J)/YF-1.)-B*(TAU(J)/YF-2.)

READ(3,FFUNC)
WRITE(6,FFUNC)

DO 20 J=1,JDIM
   WRITE(13,80) TAU(J),FTAU(J)
20 CONTINUE

WRITE(13,75)
DO 20 J=1,JDIM
   WRITE(13,80) TAU(J),FTAU(J)
20 CONTINUE

FORMAT(/2X,F8.4,F8.4,F8.4)
RETURN
END
SUBROUTINE AREA(S,PTAU,TAU,JOIN)

DIMENSION SJOIN),TAU(JOM),PTAUJOIN)

DIMENSION F(900)

C Obtain the equivalent area from F-function via Abel Transform

A(A) = \int_0^\infty \int_0^\infty \frac{f(t)}{\sqrt{t-y}} dy dt

S(1) = 0
TAU(1) = 0
DO 10 J=1,JOIN
SS = 0
DO 7 I=1,JOIN
C 7 I=1,JOIN
C SS = SS + 2 * F I
DT = TAU(I) - TAU(I-1)
F I = F I + SS
DO 6 I=1,JOIN
C 6 I=1,JOIN
C DT = TAU(I) - TAU(I-1)
C F I = F I + DT * PTAU(I) / SQRT(TAU(I) - TAU(I-1))
C CONTINUE
6 CONTINUE
7 CONTINUE
C
10 CONTINUE
C FORMAT(2E15.8,2E15.8)
FORMAT(15,15,15)
WRITE(12,15)
15 FORMAT(2F15.8,2F15.8)
DO 20 J=1,JOIN
WRITE(12,80) TAU(J),SS(J)
20 CONTINUE
80 FORMAT(2E15.8,2E15.8,2E15.8)
RETURN
END
SUBROUTINE RISETIME(FMACH,P,FH,ALT,IRISE)

An empirical method to calculate the rise time of a sonic boom

Flight test data. Good for N-wave type of signal, may be somewhat
conservative (shorter rise time).

All units used are English units!!!

FMACH = Free-stream Mach number
P = Sonic boom
PSH = Shock strength
ALT = Altitude (ft)
PS = Free-stream pressure (lb/ft2)
RISE = Rise time (sec)

DIMENSION (PH,NP)

DO 300 I=ISH+1,NP
T(I) = T(I-1) + BT
300 CONTINUE
RETURN
END

SOURCE PROGRAM

UNIX™

FORTRAN Program

SOURCE TEXT
SUBROUTINE DISTARC(X,Y,Z,NNEW,TH,IMAX,NMAX,MAX,IFLAT)

DIMENSION X(N),Y(N),TH(NNEW),IMAX(NMAX)

C This program redistributes the points (X,Y) by subroutine DISTRI
C based on the arc length. FAC is the first grid spacing. Note that
C the end points of the two sets are the same.
C
C IF(IFLAT=0), grid points will cluster near the first point, -1 near the end.
C
C Input array is (X(1),Y(1), I=1,...,N)
C Output array is (XNEW(1),TH(1)), I=1,...,NNEW
C
C PARAMETER (MAX=2000)
C
C Dimension of output array, XNEW(1), Z(NMAX), TH(MAX)
C
C Maximum number of points allowed is MAX
C
C IF(MAX.LE.0 OR MAX.LE.NNEW) THEN
C WRITE(*,'*) 'SUB DISTARC : MAX is less than N or NNEW'
C STOP ENDIF

C Look for total arc length
C
C TOTARC(1) = 0.
C DO 10 I=2,N
C ARC = SQRT( (X(I)-X(I-1))^2 + (Y(I)-Y(I-1))^2 )
C 10 TOTARC(I) = TOTARC(I-1) + ARC
C CONTINUE

C Apply subroutine DISTRI to obtain the stretching function S
C
C IF(PAC=1.0) THEN
C DELT= FAC*TOTARC(I)/FLOAT(NNEW-1)
C CALL DISTRI( DELT, NNEW, S, IFLAT)
C ELSE
C S(I) = 0.
C DO 20 K=2,NNEW
C S(I) = S(K-1) + 1./FLOAT(NNEW-1)
C 20 CONTINUE ENDIF

C Redistribute, put new array in a temporary array XN and TH
C
C XN(1)=X(I)
C TH(I-1)=TH(I)
C XN(NNEW)=X(N)
C TH(NNEW)=TH(N)
C
C DO 40 I=1,NNEW
C ARCNEW = S(I)*TOTARC(I)
C DO 50 I=1,N
C F(TOTARC(I)) EQ ARCNEW) THEN
C XN(2) = X(I)
C TH(2) = Y(I)
C 50 CONTINUE GOTO 40

C IF(ARCNEW(TOTARC(I))/ARCNEW(TOTARC(I)-1)) THEN
C 11 = X(K-1)
C 12 = X(K)
C 13 = Y(K-1)
C 14 = Y(K)
C 15 = X + (X(K)-X(K-1))* (ARCNEW(TOTARC(I)-1))/TOTARC(I)
C CALL LIMIT(11,12,13,14,15,22,23,24,25,26,27)
C XN(2) = 22
C TH(2) = 27
C 40 CONTINUE GOTO 50

C Write the temporary arrays into the output array, THEN
C
C DO 70 J=1,NNEW
C XNEW(J) = XN(2)
C THEN(J) = TH(2)
C 70 CONTINUE RETURN

END
SOURCE PROGRAM

LHF.f

SOURCE TEXT

LINE # | SOURCE TEXT
--- | ---
1077 | ********************************************************************************************************
1078 | SUBROUTINE DISTR(YANG,EPCS,F,FIME)
1079 | PARAMETER (MAX=500)
1080 | DIMENSION S(MAX),DUM(MAX)
1081 | C........Calculating the stretching function S when given
1082 | C........the first spacing, YANG, and the number of points EPCS
1083 | C........If EPC=1, distribution is clustering at outer grid
1084 | C
1085 | IF(MAX.LE.EPCS) THEN
1086 | WRITE(*,*)'SUB DISTR : MAX is less than EPCS'
1087 | STOP
1088 | ENDIF
1089 | IF(EPCS.EQ.1) THEN
1090 | S(1) = 0.
1091 | GOTO 40
1092 | ENDIF
1093 | C
1094 | DZ1 = YANG
1095 | EP = EPCS-1
1096 | DIET = 1./FLOAT(EP)
1097 | EDBETA = 1.5
1098 | CALL GEBET(DZ1,EP,C.0001,106,EDBETA)
1099 | CALL FLI(EPCS,EDBETA,DIET,S)
1100 | C
1101 | IF(EPCS.EQ.1) THEN
1102 | DO 37 K=1,EPCS
1103 | S(K) = DUM(EPCS-E+1) = S(K)
1104 | CONTINUE
1105 | DO 38 K=1,EPCS
1106 | S(K) = 1.-DUM(K)
1107 | CONTINUE
1108 | ENDIF
1109 | CONTINUE
1110 | RETURN
1111 | END

UNIXTM
FORTRAN Program
 SUBROUTINE FSL(L1,BETA,DET,Z)

 C COMPUTES NORMALIZED NORMAL DISTANCE, D(L)

 DIMENSION Z(250)
 IF(BETA.EQ.1.) THEN
  DO 10 L=1,L1
  D(L)=0.
 10 CONTINUE
 ELSE
  DO 20 L=1,L1
  ETA=(L-1)*DET
  RR=(TBETA-1.)/(TBETA-1.)
  EEE=1.-ETA
  BETA=RR*EEE
  T(L)=(TBETA-1.)*(B-BETA)/(B-BETA-1.)
 20 CONTINUE
 END IF
 RETURN
 END
SUBROUTINE GRET(DFM, NPT, TPCC, ICC, BETA)

DIMENSION T(250)

CALL F2(1, TF, LDFL, T, 0, 0.0)

DO 20 I = 1, ICC

IF (ABS(BETA) .LT. 0.1) THEN

CALL F2(2, BETA, DET, 2)

ENDIF

IF (T(I) .LT. 0.0) THEN

RETURN

ENDIF

END

END

C

SOURCE PROGRAM
LHF.f
SUBROUTINE LININT(X1,X2,Y1,Y2,XLOC,YLOCAL)
C This subroutine linearly interpolate YLOCAL when given (X1,Y1) & (X2,Y2)
IF(X1.EQ.X2) THEN
  YLOCAL=(Y2-Y1)/2.
ENDIF
SLOPE = (Y2-Y1)/(X2-X1)
YLOCAL = SLOPE*(XLOC-X2) + Y2
100 CONTINUE
RETURN
END

YLOCAL = (Y2-Y1)/2.
SUBROUTINE MARCH(NMAX, NP, Y, YSTP, F )
DIMENSION F(NP), Y(NP)
DIMENSION YSTP(NMAX)
DIMENSION INDEX(40), FS(40)
COMMON/SHOCK/ INSC

This subroutine marches the Y direction and checks if the areas are balanced and then places the shock

KOUNT = 0
YEND = Y(NP)
100 CONTINUE
DO INSCT = 1, NP
INDEX(INSCT) = 0
ENDO
CALL SBRPT(NMAX, NP, Y, YSTP, F, INDEX, FS, YS, 0)

For all shock, do need to check the possible positions of shock
IF(Y(NP) GT YSTP(NMAX) - YSTP(1)) GOTO 400

Only one possible location of shock
IF(INSC EQ 3) GOTO 400

More than one possible locations of shock
IF(INSC GE 5) THEN

YA = YS
CALL SBRPT(NMAX, NP, Y, YSTP, F, INDEX, FS, YS, 3)
GOTO 400
ELSE

There are two separated shocks
For the shock is actually locate on the turning edge of F-function we need to relocate it.

Fix the Y1 and Y2 of this small region
IF(Y(NP) GT Y(INSC + 1)) THEN
Y2 = YS
BIG = 0.
DO ITEST = INDEX(INSCT) - 1, NP
IF(Y(NP) GT Y(INSC + 1)) AND ABS(Y(NP) - Y(INSC + 1)) GT BIG THEN
BIG = ABS(Y(NP) - Y(INSC + 1))
ENDIF
IF(Y(INSC + 1) LE Y(NP)) GOTO 300
ENDO
CONTINUE

300

Find YS by bisecting Y1 and Y2
NC = 500
DO 320 IC = 1, NC
YS = 0.5*(Y2 + Y1)
CALL POINT(NP, Y, F, INDEX, FS, YS, INSCT)
DO II = INSCT, IC
IF(INDEX(II) LE ITEST) THEN
INSCT = II
GOTO 310
ENDIF
310 CONTINUE

IS = INDEX(1)
IF(INSC EQ INSCT) THEN
CALL AREA(NP, Y, F, FS, YS, IS)
IF(IFLAT EQ 0) GOTO 400
IF(IFLAT GT 0) THEN
V2 = YS
ELSE
V1 = YS
ENDIF
320 CONTINUE
WRITE(10,*) 'After', NC, 'steps of bisection'
ENDIF
GOTO 400
ENDIF
ENDIF
END

400 CONTINUE

Form the shock
IF(INSC LE 1) AND (Y(NP) GE YEND) RETURN
DECL = PS(INSC) - F(S(1))
IF(DECL EQ 0.0) THEN
WRITE(15, 100) 'SDT: ZERO DIVISION ABOUT TO HAPPEN IN MARCH'
df = 1.0/32
else
DF = DF*DECL/INDEX(1)
IF(INX EQ INDEX(INSC) - INDEX(1)) ENDIF
F(S(1)) = F(S(1)) + Y(INSC) + YS
IS = INDEX(1) + 1
DO 450 I = IS, INDEX(INSC)
Y(I) = YS
F(I) = F(I - 1) + DF
450 CONTINUE
IF(EOUNT EQ 20) THEN
WRITE(15, 100) 'EOUNT = 20'
RETURN
ELSE
EOUNT = EOUNT + 1
GOTO 100
ENDIF
END
SUBROUTINE SIMPSON(X,F,N,X0,X1,SUM)
DIMENSION X(N),F(N)
C
N = N/2
SUM = 0.
DO 10 I=1,N
10 CONTINUE
C
SUM = F(I) + SUM + 2.*F(2*I-1)
SUM = SUM + 4.*F(2*I) + 2.*F(2*I+1)
RETURN
END
Appendix B

SAMGRID (Fortran Listing)
PROGRAM SAMGRID
include 'agrid.com'

Dr. Samuel Cheung
Date: Dec., 1993
This subroutine reads a surface grid in airfoils
sections and reformates it to produce
a surface grid of axisymmetric cross-sections.

Date: Dec., 1993 Version 3.0
read input geometry

OPEN(UNIT=12, FILE='AGRID.IN', STATUS='OLD', FORM='FORMATTED')
OPEN(UNIT=16, FILE='ACGRID.IN', STATUS='OLD', FORM='FORMATTED')
OPEN(UNIT=40, FILE='WIND.IN', STATUS='OLD', FORM='FORMATTED')

NSEC = ? of sections (streamwise stations) of the new grid
NPTS = ? of pts in the circumferential direction (MUST be odd)
NPTM= max streamwise stations
FAC = the first grid spacing is DISTR
 XIII = X leading edge
 ARROWING = 0, arrow wing
 ETIP = number of points in the wing-around direction or one surface
 NC = number of cut in the spanwise direction
 NPOLE = number of points in the upper part of the wing
 NPOLE = number of points in the lower part of the wing

Read surfgrid dimensions (nsec x npts x 1)
NAMELIST /WIND/ NSEC, NPTS, FAC
READ(40, WIND)
WRITE(*, WIND)

Read the input grid

CALL WING

Setup distribution of cross-sections to be obtained (solid)
TRIP=ZB(1)
ETSK=ETE(1)-CHORD(1)
WTP=ETE(1)-CHORD(1)
IF(ETE .LT. WTP) THEN
  ET = ETW
  ARROWING = -1.
ELSE
  ET = WTP
  ARROWING = 1.
ENDIF
WAKE = AMAX(ETW, WTP)
DO 100 J=1, NSEC
DIST(J)*=ETIP+(ET-J)/ETIP(NSEC-1)
100 CONTINUE

The nose of the wing
DO 187 I=1, NPTS
  XOUT(I,E)=EDIST(1)
  YOUT(I,E)=YNEW(I)
  ZOUT(I,E)=ZDIST(NPTS-E-I)
187 CONTINUE

Begin main loop for each x-station
DO 600 J=1, NSEC
  LOCAL=EDIST(L)
  Redistribute the points from spanwise out to streamwise cut-
The output EDIST and YNEW are from the root to the tip, therefore
  The output (EDIST,YNEW) in both surfaces have EDIST # of pts in
  the circumferential direction, their last point have the same physical
  value for both surfaces.
  Do the lower surface
  CALL REDIST(LOCAL, ETIP, FAC, 1)
  DO 300 K=1, ETIP
    XOUT(L,K) = LOCAL
    YOUT(L,K) = YNEW(K)
    ZOUT(L,K) = EDIST(K)
  300 CONTINUE

  Do the upper surface
  CALL REDIST(LOCAL, ETIP, FAC, 1)
  DO 400 K=ETIP+1, NPTS
    XOUT(L,K) = LOCAL
    YOUT(L,K) = YNEW(K)
    ZOUT(L,K) = EDIST(K)
  400 CONTINUE

  For the computational grid of SPGID code
  the wake has to have different pts in same
  physical location, such that (Z1,Z2)=[Z1,Z2] Here
  the calculation divided into upper and lower parts.
  For safety sake, set Z1=Z2
  IF(LOCAL LE WAKE) GOTO 500
  DO 500 K=1, ETIP
    XOUT(L,K) = ETIP-K
  500 CONTINUE

*Singel model
* IF(ABS(YOUT(L,KI)-YOUT(L,K2)) LE .00-4) THEN
  IF(ABS(YOUT(L,K1)-YOUT(L,K2)) LE .00-2) THEN
    IF(ABS(YOUT(L,K1)-YOUT(L,K2)) LE .00-5) THEN
      ZOUT(L,K1)=YOUT(L,K2)
      ZOUT(L,K2)=YOUT(L,K1)
      ENDIF
    110 CONTINUE
  ENDIF
  120 CONTINUE

Proceed to next x coordinate

Write out new surfgrid in plotid format
14 WRITE(50;IPTS,EW,NSC)
15 DO 1334 I=1,NSC
16 WRITE(50) ;(OUT(I,xE),x=1,NPTS)
17 
18 1334 CONTINUE

Line 1: Initialize with the source program details.
Line 2: Write out original database in plotid format.
Line 3: Read the fuselage grid and combine the fuselage with the wing grid to form a whole configuration.
Line 4: Read the nacelles grid and combine the nacelles with the wing-body grid.
Line 5: Close files.
Line 6: End.
50 SUBROUTINE ADDGRID(NP11,NP21,E,L,S,H,MSEC,EDIM)
51 NP11 and NP21, and the new dimension is MSEC again
52 PARAMETER (MAX=100)
53 DIMENSION X(NP11),V(NP11),Z(NP11),T(NP11,E,L,S,H)
54 IF(MAX.LE.NP11) THEN
55 WRITE(*,*) 'SUB ADDGRID : MAX is less than NP11'
56 STOP
57 ENDIF
58 IF(NP11.GE.NP21) THEN
59 WRITE(*,*) 'No plane is added in the streamwise direction'
60 STOP
61 ENDIF
62 C
63 C Interpolating the new grid, and put it in a temporary array
64 XI = X(NP11)
65 X2 = X(NP21)
66 Y1 = Y(NP11,E)
67 Y2 = Y(NP21,E)
68 Z1 = Z(NP11,E)
69 Z2 = Z(NP21,E)
70 CALL LININT(X1,X2,Y1,Y2,Z1,Z2,Y,E)
71 YTEMP(E) = Y
72 E10 CONTINUE
73 C
74 C Re-number the later stations
75 MSEC = MSEC+1
76 DO 10 I = MSEC,MSEC+1,-1
77 X(I) = X(I-1)
78 Y(I) = Y(I-1)
79 Z(I) = Z(I-1)
80 Z(I) = Z(I-1)
81 CONTINUE
82 C
83 C Put the temporary array in the grid
84 DO 50 I = 1,EDIM
85 X(NP11) = X(I)
86 Y(NP11,E) = YTEMP(E)
87 Z(NP11,E) = ZTEMP(E)
88 50 CONTINUE
89 C
90 RETURN
91 END
SUBROUTINE CIRCLES(Xmax, Xm, Ym, Ymax, X, Y, radius, X1, X2, X3, Y1, Y2, Y3, radius2)

DIMENSION Xmax(Xmax),YM(Xmax),Xm(Xmax),Ym(Xmax),X(Xmax),Y(Xmax),radius(Xmax),X1(Xmax),X2(Xmax),X3(Xmax),Y1(Xmax),Y2(Xmax),Y3(Xmax)

! This subroutine replaces the points (Y(Y)) if greater than X by the points on the circle.

C
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
![n]
C
C SUBROUTINE SPLINE(N,NMAX, NENM, YNENM, M1)
C
C PARAMETER (NMAX=100)
C REAL X(N), Y(N), ZENM(NENM), YNENM(NENM)
C
C THIS SUBROUTINE PRODUCES A MONOTONE CUBIC SPLINE INTERPOLANT
C TO THE DATA (X(I),Y(I)) I=1,...,N AND COMPUTES VALUES AT
C THE NEW POINTS YNENM(I), I=1,...,NENM. THESE ARE RETURNED IN
C ARRAYS YNENM(I). THE ALGORITHM USED IS THAT OUTLINED BY FRIEDEL
C
C WRITTEN BY JEFF CORDONA 10/15/86
C
C REAL D(NMAX), DEL(NMAX), R(NMAX)
C
C END
C
C MESH SPACING AND FIRST DIVIDED DIFFERENCE
C
DO 10 I=1,N-1
10 XI = X(I+1) - X(I)
100 CONTINUE

C DO 200 I=1,N-1
200 DEL(I) = (Y(I+1) - Y(I)) / XI
200 CONTINUE

C SPLINE COEFFICIENTS
C
C *** LINEAR INTERPOLATION FOR N=2 CASE ***
317 IF (N.EQ.2) THEN
318 DEL(1) = DEL(1)
319 D(1) = DEL(1)
320 GO TO 339
319 ENDIF

C *** MONOTONE SPLINE COEFFICIENTS FOR N > 2 CASE ***
339 BSNM = X(1) + X(2)
340 W1 = (X(1) - BSNM) / BSNM
340 W2 = (X(2) - BSNM) / BSNM
341 D(1) = W1*DEL(1) + W2*DEL(2)
342 IF (PCBST(DEL(1),DEL(2)) .LE. 0.) THEN
343 DEL(1) = 0.
344 ELSEIF (PCBST(DEL(1),DEL(2)) .LT. 0.) THEN
345 DMAX = 3.*DEL(1)
346 IF (ABS(DEL(1)) .GT. ABS(DMAX)) D(1) = DMAX
347 ENDIF

C INTERIOR POINTS (BILLIGE MODIFICATION OF BULYAN FORMULA)
C
350 CONST = 1. / 3.
350 DO 360 I=2,N-1
351 TOP = DEL(I-1) + DEL(I)
352 TOP = TOP + 5. * (1. + SIGN(1.,TOP))
353 ALPHA = CONST * ((1. + W1) / (W1-I+1) + I(I))
354 BOT = ALPHA - DEL(I) + (1.-ALPHA) * DEL(1-I) + I-I-20
355 D(1) = TOP / BOT
356 D(1) = D(1)
357 360 CONTINUE

C LAST BOUNDARY POINT (USE THREE POINT FORMULA ADJUSTED TO BE
C SHAPE PRESERVING)
C
368 BSNM = X(N-2) + X(N-1)
369 W1 = (X(N-1) - BSNM) / BSNM
370 W2 = (X(N) - BSNM) / BSNM
371 D(N) = W1*DEL(N-2) + W2*DEL(N-1)
372 IF (PCBST(DEL(N-1),DEL(N)) .LE. 0.) THEN
373 D(N) = 0.
374 ELSEIF (PCBST(DEL(N-2),DEL(N-1)) .LT. 0.) THEN
375 DMAX = 3.*DEL(N-1)
376 IF (ABS(DEL(N)) .GT. ABS(DMAX)) D(N) = DMAX
377 ENDIF

C RETURN POINTS (BILLIGE MODIFICATION OF BULYAN FORMULA)
C
385 DMAX = Y(N-1) + Y(N)
386 D(1) = DMAX
387 390 CONTINUE

C SPLINE EVALUATION
C
390 KENM = I.E. 0(N)
390 DND = 0. + 2*NKENM + 3
390 DO 400 J=1,N-1
400 CYNENM = (D(J) + D(J+1) + 2.*DEL(J)) / (H(J)+H(J+1))
400 CTWO = (3.*DEL(J) + 2.*D(J) - D(J+1)) / H(J)
400 CRAY = IENM(J) + ISRCBRK(NENM,ENEM,,X(N),X(N+1)) 100 CRAY
400 J = XENM(J) + X(N)
400 T = ZENM(J) - X(J)
400 T = T + T*(D(J) + T*(CTWO + T*CTREE))
400 400 CONTINUE

C
C RETURN
C
412 END
FUNCTION  PCBST(ARG1,ARG2)

C

PCBST = SIGN(1.,ARG1) * SIGN(1.,ARG2)

IF ((ARG1.EQ.0.) .OR. (ARG2.EQ.0.)) PCBST = 0.

C

RETURN

END
FUNCTION ISRCHGE(N,X,INCX,F_TARGET)
DIMENSION X(*)
IF(N.LE.0) THEN
ISRCHGE = 0
RETURN
ELSE
IT = 1 + (N-1) * INCX
ISRCHGE = 1
DO 10 I=1,IT,INCX
IF(X(I).GE.F_TARGET) GOTO 11
CONTINUE
CONTINUE
11 CONTINUE
10 CONTINUE
ENDIF
RETURN
END
**SUBROUTINE CURT**

**DIMENSION** TKW(NFI), IWL(NFI)

**LJ = L - 1**

**NFUS = EDIM - NPTS**

**NROT = NFUS/2 + 1**

**NTOP = NFUS/2 + 1**

**C**

**DO 990 LL=1,LJ**

**Note: I am leaving the nose and the wake alone**

**IF (ABS(Y(LL, NROT)-Y(LL, EDIM-NTOP+1)) .LE. 1.E-7) THEN**

**RETURN**

**END**

**C**

**Do the bottom first:**

**DO 10 K=1, NROT**

**ZINT(K) = Y(LL,K)**

**CONTINUE**

**10**

**FGPS = SQRT((Z(LL,NROT)-Z(LL,NROT+1))**2** +**

**((Y(LL,NROT)-Y(LL,NROT+1))**2**)**

**CALL DISTARC(ZINT, ZINT, NROT, TWK, ZINT, NROT, FGPS, 1)**

**DO 80 K=1, NROT**

**T(L,L) = TWK(K)**

**L(L,L) = Y(NK)**

**80 CONTINUE**

**C**

**Do the top now:**

**DO 100 K=1, NTOP**

**VINT(K) = Y(LL,K+(EDIM-NTOP))**

**ZINT(K) = Z(LL,K+(EDIM-NTOP))**

**CONTINUE**

**100**

**W1 = (EDIM-NTOP)**

**W2 = (EDIM-NTOP)-1**

**FGPS = SQRT((Z(LL,NROT)-Z(LL,W1))**2** +**

**((Y(LL,NROT)-Y(LL,W1))**2**)**

**CALL DISTARC(VINT, ZINT, NROT, TWK, ZINT, NROT, FGPS, 0)**

**DO 180 K=1, NTOP**

**T(L,L) = TWK(K)**

**L(L,L) = Y(NK)**

**180 CONTINUE**

**990 CONTINUE**

**RETURN**

**END**
SUBROUTINE DISTARC(X,Y,N,XNEW,YNEW,PPS,FILAT)
C
DIMENSION X(N),Y(N),XNEW(NNEW),YNEW(NNEW)
C
This program redistributes the points (x,y) by subroutine DISTRI
C
based on the arc length. PPS is the first grid spacing. Note that
C
the end points of the two sets are the same.
C
IFILAT=0, grid points will cluster near the first point, -1 near the end.
C
Input array is (X(1),Y(1)), (1),...., N
C
Output array is (XNEW(1),YNEW(1)), (1),...., NNEW
C
PARAMETER (MAX = 40)
C
DIMENSION 1(MAX),TORTAC(MAX),K(MAX),TH(MAX)
C
Maximum number of points allowed is MAX
C
IF (MAX.LE.N .OR. MAX.LE.NNEW) THEN
WRITE(*,'*)) 'SUB DISTARC : MAX is less than N or NNEW'
STOP
ENDIF
C
Look for total arc length
TORTAC(1) = 0.
DO 10 K=2,N
ARC = SQRT(X(K)-X(K-1))**2 + (Y(K)-Y(K-1))**2
TORTAC(K) = TORTAC(K-1) + ARC
10 CONTINUE
C
Apply subroutine DISTRI to obtain the stretching function A
C
For PPS = 0, equal spacing is used
IF (PPS = 0.) THEN
DELT = PPS/TORTAC(N)
ELSE
DO 25 K=1,NNEW
S(K) = S(K-1) + 1./FLOAT(NNEW-1)
25 CONTINUE
ENDIF
C
Redistribution, put new array in a temporary arrays XN and YN
EN(1)=X(1)
YN(1)=Y(1)
EN(NNEW)=X(N)
YN(NNEW)=Y(N)
C
DO 60 J = 2,NNEW
ARCHEN = S(J)/TORTAC(N)
DO 55 K = 2,N
IF (ABS(TORTAC(K)-ARCHEN).LE.1.E-7) THEN
XN(K) = X(K)
YN(K) = Y(K)
GOTO 60
ENDIF
55 WRITE(*,'()))) 'SUB DISTARC : ARCNEW not in interval
ELSE
IF (TORTAC(K).GT.ARCHEN) THEN
X1 = X(K-1)
X2 = X(K)
Y1 = Y(K-1)
Y2 = Y(K)
XX = (X(K)-X(K-1))**2 + (Y(K)-Y(K-1))**2
6 (ARCHEN-ARCHEN)/X1.Y2.Y1.X2.Dhift(ARCHEN-ARCHEN)
CALL LINTP(X1,X2,Y1,Y2,XX,Y)
ENDIF
ENDIF
54 CONTINUE
60 CONTINUE
C
Write the temporary arrays into the output XNEW, YNEW
DO 70 J=1,NNEW
XNEW(J) = XN(J)
YN(J) = YN(J)
70 CONTINUE
RETURN
END
SUBROUTINE DISTRI(FANG, EPCS, S, IFINE)

PARAMETER (MAX=400)

DIMENSION S(MAX), EOM(MAX)

C......Calculating the stretching function & when given
C......the first spacing, FANG, and the number of points EPCS
C......If IFINE=1, distribution is equalizing at outer grid

IF(MAX LE EPCS) THEN
    WRITE(*,*)'SUB-DISTRI: MAX is less than EPCS'
    STOP
ENDIF

IF(EPCS EQ 1) THEN
    S(1) = 0.
    GOTO 40
ENDIF

DI1 = FANG
EFM = EPCS-1
DELTA = 1./FLOAT(EFM)
RDBETA = 1.5
CALL GRBET(DI1, EFM, 0.0001, 100, RDBETA)
CALL FZ1(EPCS, RDBETA, DI1, S)

DO 37 K=1,EPCS
    DUM(KFCS-K+I) = S(K)
7 CONTINUE

DO 38 K=E_E_PCS
    S(K) = 1.-DUM(K)
8 CONTINUE

ENDF

40 CONTINUE
RETURN
END
SUBROUTINE ED_E(NC, NU, NL, XL, XBK, XBAS, YBASE, ZBASE, NPK, LS)
DIMENSION XBAS(NP, 2, LS), YBASE(NP, 2, LS), ZBASE(NP, 2, LS)

JX = ZBASE(1, 1, 1)

DO 200 K = 1, NC
IF (XBAS(1, 1, K) GT XBK) THEN
   X1 = XBAS(1, 1, K-1)
   X2 = XBAS(1, 1, K)
   Z1 = ZBASE(1, 1, K-1)
   Z2 = ZBASE(1, 1, K)
   CALL LININT(X1, X2, Z1, Z2, XBK, ZBK)
   GOTO 210
ENDIF
200 CONTINUE
210 CONTINUE

IF (BASE(1, 1, E) .GT. ZB[) GO TO 700
CALL LININT(Z_E, XBK, XL, XB_E, XI, J_E(1, 1, _), XL_)
XL_OLD = X_SE(I, I, _)
XLT = XBASE(NL, I, K)
DO 280 I = 1, NL
   IF (XBAS(I, I, K) .LE. XL_OLD) THEN
      E = XLT-XBASE(I, 1, K)
      XBAS(I, 1, K) = (F*XTL + E*XL_)/(F+E)
   ENDIF
280 CONTINUE

200 CONTINUE

IF (Z_E .GT. XBASE(NU, 1, E)) GO TO 700
CALL LININT(Z_E, XBASE(NU, 1, E), XL, XBASE(1, 2, E))

300 CONTINUE
500 CONTINUE
700 CONTINUE
RETURN
END
SUBROUTINE EQSPACE

include "sgrid.com"

L2 = L-1
DO 130 I=1,L2
   X(2L) = X(I)
   DX = DX/FL(2L)
   DO 120 J=I+1,L2
      X(J) = X(J-1)+DX
      IF(ABS(X(J)-X(J-1)) .LE. 1.E-7) THEN
         CONTINUE
      END IF
      IF(X(J) .GT. X(J-1)) THEN
         DO 145 K=I,KDIM
            X1 = X(K)
            Y1 = Y(IN(K))
            Z1 = Z(IN(K))
            X2 = X(K+1)
            Y2 = Y(IN(K+1))
            Z2 = Z(IN(K+1))
            CALL LININT(X1,X2,Y1,Y2,XX,YY)
            CALL LININT(X1,X2,Z1,Z2,XX,ZZ)
            CONTINUE
         END IF
      END IF
   END
120 CONTINUE
130 CONTINUE
135 CONTINUE
140 CONTINUE
145 CONTINUE
GOTO 160
150 CONTINUE
160 CONTINUE
RETURN
END
SUBROUTINE FILE(Y,1,EDIM,M1,M2,MT1,MT2,RF1)

PARAMETER (M1=400)
DIMENSION Z(EDIM),Y(EDIM)
DIMENSION DI(MAX),D2(MAX)
COMMON /EXP/,EDIM,MT1,MT2

687 C This subroutine takes a wing-fuselage shape, \((Y(k),Z(k))\) \(k=1,EDIM\),
688 C and finds the two (top and bottom) intersections of the wing and the
689 C fuselage.
690 C And then, for example, at the bottom intersection \((Y(1),Z(1))\), it
691 C extends to a segment of points \((Y(k),Z(k))\) \(k=1\) to \(k2\), where
692 C \(Y(k)<RF1\), \(Y(k2)>RF1\).
693 C Similar procedure for the top part.
694 C and then, call subroutine CIRCLE to replace the segment by a segment
695 C of a circle with radius RF1.
696 C
697 C IF(RF1.EQ.0.) GOTO 735
698 C IF(MAX.LE.EDIM) THEN
699 C WRITE(*,*)'FILE : MAX is less than EDIM'
700 C END
701 C
702 C Bottom part of the aircraft
703 C IF(M1.EQ.0. AND. M2.EQ.0.) GOTO 135
704 DO 130 K=1,EDIM
705 IF (X.EQ.MT1 .AND. Z.GT.ZROOT) THEN
706 E1=E-M1
707 E2=E-M2
708 CALL CIRCLE(E1,E2,EDIM,Y,Z,RF1,ARCORS)
709 N=E-EF1+1
710 CALL DISTARC(D2,DI,N,D2,DI_N,-10.,0)
711 DO 120 KD-KFI,KF2
712 DI(KD-KFI+I) = D2(KD)
713 D2(KD-KFI+I) = Z(KD)
714 CONTINUE
715 C
716 C Top part of the aircraft
717 C IF(M1.EQ.0. AND. M2.EQ.0.) GOTO 735
718 DO 700 X=MT2,EDIM
719 IF (Y.EQ.MT2 .AND. Z.LE.ZROOT) THEN
720 E1=E-MT2
721 E2=E-M1
722 CALL CIRCLE(E1,E2,EDIM,Y,Z,RF1,ARCORS)
723 N=E-EF1+1
724 CALL DISTARC(D1,D2,N,D1,D2_N,10.0)
725 DO 530 KD-KFI,KF2
726 DI(KD-KFI+I) = D1(KD)
727 D1(KD-KFI+I) = Y(KD)
728 CONTINUE
729 C
730 C RETURN
731 END
SUBROUTINE PZI(LI, TBETA, DET, Z)

! COMPUTES NORMALIZED NORMAL DISTANCE, Z(L)

PARAMETER (MAX=400)
DIMENSION Z(MAX)

IF(MAX.LE.LI) THEN
  WRITE(*,*) 'SUB PZ1: MAX is less than L1'
  STOP
ENDIF

IF(TBETA.EQ.0.) THEN
  DO 10 L=1,LI
    Z(L)=0.
  CONTINUE
  ELSE
    ETA=(L-1.)*DET
    RR=(TBETA-1.)/(TBETA-1.)
    EEG=1.-ETA
    RRDR=BETA+RR+EE
    Z(L)=TBETA*EE/(TBETA-1.)
    CONTINUE
    END IF
END
SUBROUTINE GRET (DFM, NPT, FPCC, IICC, BETA)

! BISECTION METHOD USED TO DETERMINE STRETCHING PARAMETER, BETA, WHICH GIVES DESIRED GV AT THE WALL

PARAMETER (MAX=400)
DIMENSION Z(MAX)
WRITE ('**','**') 'SUB GRET : MAX is less than NPT'
STOP
ENDIF

IICC=ICC
FPCC=FPCC*DFM
BETA=BETA
Z1=DFM
DET=1./NPT
NB=1
FP=1.5
ITCC=ICC/10
GO TO 1,1,1CC
IF(RT.GT.Z0) THEN
RETURN
END SUBROUTINE GRET
SUBROUTINE LINTINT(X1,X2,Y1,Y2,LOCAL,YLOCAL)
C This subroutine linearly interpolate YLOCAL when given (X1,Y1) & (X2,Y2)
IF((ABS(X1-X2) LE 1.E-7)) THEN
  YLOCAL=(Y2*Y1)/2.
ENDIF
SLOPE = (Y2-Y1)/(X2-X1)
LOCAL = SLOPE*(XLOCAL-X2) + Y2
100 CONTINUE
RETURN
END
SUBROUTINE MONT(JN,LNUM,IPRT)
INCLUDE "MID10.COM"
DIMENSION NPAIR(2*NPI),LNUM(4)
DIMENSION NPAIR(2*NPI),TANAC(NPI),ZENG(NPI)
COMMON /ENG/, XENG(2,NPI),YENG(2,NPI),ZENG(2,NPI)
COMMON /NAC2/, NAC(2),NACP(2)

MC = 40
HNP = NPTS + MC
If(LUN1(1).EQ.LNUM(3)) .AND. LNUM(2).EQ.LNUM(4) THEN
NPTS = NPTS + MC
HNP = NPTS + MC
ENDIF
NPAIR(1,1) = NPTS + 1
HNP = (NPTS+1)/2
ENDIF
NAC = NAC(JN)
NAC = NAC(JN)
IF(IPRT.NE.0) WRITE(I1UNN),NPTS,LNUM(2)-LNUM(1)+1
ENDIF
DO 100 L = LNUM(1),LNUM(2)
CALL KDI(L),NPAIR(1)
100 CONTINUE
END

C Integrate the points of the local at MOUT
DO 100 C = 1,NHAC
IFABS(XENG(JN,LN)-XOUT(L,1)).LE.1.E-7 THEN
DO 2 = 1,NHAC
IFABS(YENG(JN,LN)-YOUT(L,1)).LE.1.E-7 THEN
ENDIF
ENDIF
GOTO 105
105 CONTINUE
RETURN
END

C Count the number of points in the wake if we are in the wake
KOUNT = 0
DO 120 K = 1,NHAC
ZOUT(L,NPAIR(1,K)) - NOUT(L,NPAIR(1,K)) )
ENDDO
120 CONTINUE
RETURN
END
CONTINUE
984 IF(INTRAIL.EQ.2) THEN
985 DO 200 K=1,ENOUT
986 IF(ZOUT, I, NPAIR (1,E) ) LT ZNAC (NE) THEN
987 ENDI
988 DO 200 K=1,ENOUT
989 IF(ZOUT, I, NPAIR (1,E) ) LT ZNAC (NE) THEN
990 GOTO 190
991 ENDI
992 GOTO 215
993 ENDI
994 200 CONTINUE
995 CONTINUE
996 NE = NTEMP
997 IF(ZOUT, I, NPAIR (1,E) ) THEN
998 CALL LIMIT (YAC(NE-1), YAC(NE), ZNAC(NE-1), ZNAC(NE) )
999 ZOUT ( I, NPAIR ( 1,E) ) = ZOUT ( I, NPAIR ( 1,E) )
1000 ELSE
1001 CALL LIMIT ( ZOUT ( I, NPAIR ( 1,E) ) , ZOUT ( I, NPAIR ( 1,E) +1) , ZNAC ( NE-1), YTE)
1002 ZIE = ZNAC ( NE )
1003 ENDI
1004 ELSE DO 230 K=1,NPE
1005 IF (SOFT ( I, E ) GT ZNAC ( NTEMP ) ) THEN
1006 YF = 0
1007 IF (SOFT ( I, E ) LT ZNAC ( NTEMP ) ) THEN
1008 NTEMP = NTEMP - 1
1009 GOTO 190
1010 ENDI
1011 GOTO 235
1012 ENDI
1013 235 CONTINUE
1014 CONTINUE
1015 NE = NTEMP
1016 CALL LIMIT ( ZOUT ( I, E ) , ZOUT ( I, E +1) , ZOUT ( I, E) , YOUT ( E) , ZNAC ( NE), YTE)
1017 ZIE = ZNAC ( NE )
1018 ENDI
1019 Form the lower surface
1020 Store wingroot grid
1021 DO K=1,NPAIR ( 1,E)
1022 YMG ( K ) = YOUT ( K, E)
1023 ZMG ( K ) = ZOUT ( K, E)
1024 ENDDO
1025 Store wingtip grid to a temp array
1026 IF(INTRAIL.EQ.2) THEN
1027 DO K=1,NPAIR ( 1,E),NPE
1028 IF (SOFT ( I, E ) LT ZNAC ( NTEMP ) ) THEN
1029 NTEMP = NTEMP - 1
1030 GOTO 190
1031 ENDI
1032 ELSE
1033 DO K=1,NPE
1034 YINT ( K-E) = YOUT ( K, E)
1035 YOUT ( K-E) = YOUT ( K, E)
1036 ENDDO
1037 ENDDO
1038 Get the macelle grid under the wake line ready
1039 INDC = NE-NE-1+2
1040 YKE ( I ) = YTS
1041 ZKE ( I ) = ZTS
1042 YKE ( INDC ) = YTE
1043 ZKE ( INDC ) = ZIE
1044 DO K=1,NE
1045 YKE ( K-NE-2) = YKE ( K-NE-2)
1046 ZKE ( K-NE-2) = ZKE ( K-NE-2)
1047 ENDDO
1048 IF(INTRAIL.EQ.2) THEN
1049 YKE ( K-NE-2) = YKE ( K-NE-2)
1050 ZKE ( K-NE-2) = YKE ( K-NE-2)
1051 ELSE
1052 YKE ( K-NE-2) = ZKE ( K-NE-2)
1053 ZKE ( K-NE-2) = YKE ( K-NE-2)
1054 ENDDO
1055 Stick the macelle grid under the wing
1056 DO K=1,NNACC
1057 YMG ( NPAIR ( 1,E) ) = YKE ( K )
1058 ZMG ( NPAIR ( 1,E) ) = ZKE ( K-NE-2)
1059 ENDDO
1060 Pull back the wingtip grid into the wing
1061 IF(INTRAIL.EQ.2) THEN
1062 KEEP = NPAIR ( 1,E) +1
1063 DO K=1,KEEP
1064 ZMG ( NPAIR ( 1,E) +NMAC+ E ) = ZINT ( K )
1065 YMG ( NPAIR ( 1,E) +NMAC+ E ) = YINT ( K )
1066 ENDDO
1067 ELSE
1068 KEEP = NPAIR ( 1,E) +1
1069 DO K=1,KEEP
1070 ZMG ( NPAIR ( 1,E) +NMAC+ E ) = ZINT ( K )
1071 YMG ( NPAIR ( 1,E) +NMAC+ E ) = YINT ( K )
1072 ENDDO
1073 ENDDO
1074 Form the upper surface
1075 Store the wingtip grid
1076 IF(INTRAIL.EQ.2) THEN
1077 DO K=1,NPAIR (2,E)
1078 YMG ( NNACH+E-NPE ) = YOUT ( K, E)
1079 ZMG ( NNACH+E-NPE ) = ZOUT ( K, E)
1080 ENDDO
1081 NN = NNACH+ NPAIR ( 2,E) -NNPE
1082 ELSE
1083 DO K=1,NPE
1084 YMG ( NNACH+E-NPE ) = YOUT ( K, E)
1085 ZMG ( NNACH+E-NPE ) = ZOUT ( K, E)
1086 ENDDO
1087 NN = NNACH+ E-NNPE
1088 260 CONTINUE
1089 DO K=NPE,ENL
1090 YMG ( NNACH+E-NPE ) = YOUT ( K, E)
1091 ZMG ( NNACH+E-NPE ) = ZOUT ( K, E)
1092 ENDDO
C s = 1
C s = 1
C s = 1
C s = 1
C s = 1
C s = 1
C s = 1
C s = 1
C s = 1
DO 460 K=NP2,NPTS
   IF(YOUT(L,K).LE.ZNAC(NFRIS)) THEN
      NN2 = K
      CALL LININT(ZOUT(L,NN2),ZOUT(L,NN2-1),YOUT(L,NN2),
      YOUT(L,NN2-1),ZNAC(NFRIS),YY2)
      ZZ2 = ZNAC(NFRIS)
      GOTO 605
   ENDIF
   600 CONTINUE
C Store the wingtip grid
DO 620 E=NPHB,NNI
   YWNG(NPNNH÷K-NPH) = YOUT(L,K)
   ZWNG(NPWNH+K-NPH) - ZOUT(L,K)
   620 CONTINUE
   N2 = NN2-NNI+1
   YK(I) = YYI
   ZHK(N2) = ZZ2
   DO K=NNI+1,NN2-1
      YNX(K) = YOUT(L,K)
      ZWE(K) = ZOUT(L,E)
   ENDDO
   NNT = NPNNB-(NN1-NPH)-(NPTS-NN2+I)
   CALL DISTARC(YNK, ZWK, N2 ,TNK, ZNK,NN&, -10,0)
   NNT = NPNNI+(NNI-NPH)
   DO K=1,NNA
      YWNG(NNT+K) = YNK(K)
      ZWNG(NNT+K) = ZNK(K)
   ENDDO
   DO K=NN2,NPTS
      Y_G(N_+NNA+X-N_+I) = YOUT(L,E)
      Z_G(N_+NNA+K-NN2+I) = ZOUT(L,E)
   ENDDO
   700 CONTINUE
C Store the wingroot grid
IF(IPRNT.NE.0)THEN
   WRITE(IPRNT) (XOUT(L,E), E=1,NPPH),
   (YOUT(L,E), E=1,NPPH),
   (ZOUT(L,E), E=1,NPPH)
   CALL FLUSR(IPRNT)
   ENDIF
   DO 750 E=1,NPPH
      XOUT(L,E) = XOOT(_,I)
      YOUT(L,E) = YI_NG(_)
      ZOUT(L,E) = ZI_NG(_)
   750 CONTINUE
C IF(L.Num(1),EQ.L.Num(3) .AND. L.Num(2),EQ.L.Num(4)) THEN
   NPTS = NPTS - MC
   800 CONTINUE
IF(L.Num(1),EQ.L.Num(3) .AND. L.Num(2),EQ.L.Num(4)) THEN
   NPTS = NPTS - MC
   ENDIF
   RETURN
END
**SOURCE PROGRAM**

```
1260 SUBROUTINE MACGRID
1261 include "samgrid.com"
1262 c This subroutine read the nacelle grid and combine it with the wing grid.
1263 c
1264 COMMON /ENG/ XENG(2, NP1), YENG(2, NP1), ZENG(2, NP1, NP1)
1265 COMMON /EMAC/ XMAC(1, NP1), YMAC(1, NP1, NP1), ZMAC(1, NP1, NP1)
1266 * * *
1267 COMMON /MACP/ MMACP, NNACP, LNACP, LMACP
1268 * * *
1269 COMMON /MAC/ XM, YM, ZM, NM, VM, WM, DM
1270 COMMON /M/ MM, NM, LM, NM, NAD, LAD, VAD, WAD, DAD
1271 COMMON /RA/ XRA, YRA, ZRA, NRA, VRA, WRA, DRA, RA
1272 COMMON /GA/ XGA, YGA, ZGA, NGA, VGA, WGA, DAG, GGA
1273 COMMON /IA/ XIA, YIA, ZIA, NIA, VIA, WIA, DIA, IIA
1274 COMMON /PA/ XPA, YPA, ZPA, NPA, VPA, WPA, DPA, PPA
1275 COMMON /LA/ XLA, YLA, ZLA, MLA, VLA, WLA, DAL, LLA
1276 COMMON /DA/ XDA, YDA, ZDA, NDA, VDA, WDA, DDA, ADA
1277 COMMON /AA/ XAA, YAA, ZAA, NAA, VAA, WAA, DAA, AAA
1278 COMMON /GA/ XGA, YGA, ZGA, NGA, VGA, WGA, DAG, GGA
1279 COMMON /IA/ XIA, YIA, ZIA, NIA, VIA, WIA, DIA, IIA
1280 COMMON /PA/ XPA, YPA, ZPA, NPA, VPA, WPA, DPA, PPA
1281 COMMON /LA/ XLA, YLA, ZLA, MLA, VLA, WLA, DAL, LLA
1282 COMMON /DA/ XDA, YDA, ZDA, NDA, VDA, WDA, DDA, ADA
1283 COMMON /AA/ XAA, YAA, ZAA, NAA, VAA, WAA, DAA, AAA
1284 COMMON /GA/ XGA, YGA, ZGA, NGA, VGA, WGA, DAG, GGA
1285 COMMON /IA/ XIA, YIA, ZIA, NIA, VIA, WIA, DIA, IIA
1286 COMMON /PA/ XPA, YPA, ZPA, NPA, VPA, WPA, DPA, PPA
1287 COMMON /LA/ XLA, YLA, ZLA, MLA, VLA, WLA, DAL, LLA
1288 COMMON /DA/ XDA, YDA, ZDA, NDA, VDA, WDA, DDA, ADA
1289 COMMON /AA/ XAA, YAA, ZAA, NAA, VAA, WAA, DAA, AAA
1290 COMMON /MACP/ MMACP, NNACP, LNACP, LMACP
1291 * * *
1292 COMMON /MAC/ XM, YM, ZM, NM, VM, WM, DM
1293 COMMON /M/ MM, NM, LM, NM, NAD, LAD, VAD, WAD, DAD
1294 COMMON /RA/ XRA, YRA, ZRA, NRA, VRA, WRA, DRA, RA
1295 COMMON /GA/ XGA, YGA, ZGA, NGA, VGA, WGA, DAG, GGA
1296 COMMON /IA/ XIA, YIA, ZIA, NIA, VIA, WIA, DIA, IIA
1297 COMMON /PA/ XPA, YPA, ZPA, NPA, VPA, WPA, DPA, PPA
1298 COMMON /LA/ XLA, YLA, ZLA, MLA, VLA, WLA, DAL, LLA
1299 COMMON /DA/ XDA, YDA, ZDA, NDA, VDA, WDA, DDA, ADA
1300 COMMON /AA/ XAA, YAA, ZAA, NAA, VAA, WAA, DAA, AAA
1301 COMMON /GA/ XGA, YGA, ZGA, NGA, VGA, WGA, DAG, GGA
1302 COMMON /IA/ XIA, YIA, ZIA, NIA, VIA, WIA, DIA, IIA
1303 COMMON /PA/ XPA, YPA, ZPA, NPA, VPA, WPA, DPA, PPA
1304 COMMON /LA/ XLA, YLA, ZLA, MLA, VLA, WLA, DAL, LLA
1305 COMMON /DA/ XDA, YDA, ZDA, NDA, VDA, WDA, DDA, ADA
1306 COMMON /AA/ XAA, YAA, ZAA, NAA, VAA, WAA, DAA, AAA
1307 COMMON /GA/ XGA, YGA, ZGA, NGA, VGA, WGA, DAG, GGA
1308 COMMON /IA/ XIA, YIA, ZIA, NIA, VIA, WIA, DIA, IIA
1309 COMMON /PA/ XPA, YPA, ZPA, NPA, VPA, WPA, DPA, PPA
1310 COMMON /LA/ XLA, YLA, ZLA, MLA, VLA, WLA, DAL, LLA
1311 COMMON /DA/ XDA, YDA, ZDA, NDA, VDA, WDA, DDA, ADA
1312 COMMON /AA/ XAA, YAA, ZAA, NAA, VAA, WAA, DAA, AAA
1313 COMMON /MACP/ MMACP, NNACP, LNACP, LMACP
1291 c Read the nacelles geometry:
1292 c Inner nacelle geometry
1293 c CALL MACIN
1294 XIN1 = XMAC(1)
1295 XIN2 = XMAC(NMAC)
1296 ZMAC(I) = ZMAC(I)
1297 NNACP(1) = NNACP
1298 DO 100 L = 1, NNACP
1299 XENG(1, L) = XMAC(L)
1300 YENG(L, 1, E) = YMAC(L, E)
1301 ZENG(1, L, K) = ZMAC(L, K)
1302 END
1303 CONTINUE
1304 c Outer nacelle geometry
1305 c CALL MACIN
1306 XOUT1 = XMAC(1)
1307 XOUT2 = XMAC(NMAC)
1308 YMAC(I) = YMAC(I)
1309 NNACP(2) = NNACP
1310 DO 200 L = 1, NNACP
1311 XENG(1, L) = XMAC(L)
1312 YENG(L, 1, E) = YMAC(L, E)
1313 ZENG(1, L, K) = ZMAC(L, K)
1314 END
1315 CONTINUE
1316 c In a case of 1 nacelle, three zone will be made.
1317 c MENG : Station(s) will be added to the wing at inlet and outlet of the nacelle
1318 c MOUNT : Mount the nacelle under the wing and/or wake line
1319 c JN = 1 Inner nacelle
1320 c JN = 2 Outer Nacelle
1321 c The first zone, only one nacelle appears
1322 WRITE(*, *) 'zone 1'
1323 JN = 1
1324 CALL MENG(JN, XIN1, XOUT1)
1325 CALL MOUNT(JN, XIN1, XOUT1, 21)
1326 LNUN(1) = LNUN(2)
1327 LNUN(4) = LNUN(2)
1328 c The second zone consists two nacelles appear
1329 WRITE(*, *) 'zone 2'
1330 JN = 1
1331 CALL MENG(JN, XIN1, XOUT1, XOUT2)
1332 LNUN(1) = LNUN(1) + 1
1333 CALL MOUNT(JN, XIN1, XOUT1, 0)
1334 LNUN(3) = LNUN(3)
1335 LNUN(4) = LNUN(3)
1336 WRITE(*, *) 'zone 2'
1337 CALL MENG(JN, XIN1, XOUT1, XOUT2)
1338 LNUN(1) = LNUN(1) + 1
1339 JN = 2
1340 CALL MOUNT(JN, XIN1, XOUT2, 22)
1341 LNUN(1) = LNUN(1)
1342 LNUN(4) = LNUN(1)
1343 c The third zone, only one nacelle appears
1344 WRITE(*, *) 'zone 3'
1345 JN = 1
1346 CALL MENG(JN, XIN1, XOUT2)
1347 LNUN(1) = LNUN(1) + 1
1348 CALL MOUNT(JN, XIN1, XOUT2, 23)
1349 RETURN
1350 END
```
SUBROUTINE MERGING(LROW,XX,XY)
  include "agrld.com"
  DIMENSION XWNG(NPI,NPI),YWNG(NPI,NPI),ZWNG(NPI,NPI)
  DIMENSION INUM(4)
  C Rewrite the coordinate of the wing
  DO 10 I=1,NSEC
    XOUT(I) = XWNG(L,I)
    YOUT(I) = YWNG(L,I)
    ZOUT(I) = ZWNG(L,I)
  10 CONTINUE
  C Find out where the x-location of start and end of nsec
  IFR(I) = IFR(I-1)
  IE(I) = IE(I-1)
  C Add two station in the wing, these two stations lie exactly on
  C INSTR and ENEND.
  DO 20 N = 1,NN
    XWNG(N,1) = XWNG(N,NN)
    YWNG(N,1) = YWNG(N,NN)
    ZWNG(N,1) = ZWNG(N,NN)
    20 CONTINUE
  C Create an extra station in the wing
  IF((XWNG(LW,1) < X) .OR. (XWNG(LW,1) > X)) THEN
    X(1) = X(2)
    X(N) = X(1)
    Y(1) = Y(1)
    Y(N) = Y(N)
    Z(1) = Z(1)
    Z(N) = Z(N)
    CALL LINPNT(LX,LY,LZ)
    CALL LINPNT(LX,LY,LZ)
    30 CONTINUE
  ELSE IF(I .NE. 1) THEN
    X(2) = X(1)
    X(N) = X(N)
    Y(2) = Y(1)
    Y(N) = Y(N)
    Z(2) = Z(1)
    Z(N) = Z(N)
    CALL LINPNT(LX,LY,LZ)
    CALL LINPNT(LX,LY,LZ)
    30 CONTINUE
  ENDIF
  C If we are in wake, make sure top pts intersect the bottom pts
  DO 24 I = 3,(NN-1),2
    X2 = X(I)
    Y2 = Y(I)
    Z2 = Z(I)
    24 CONTINUE
  C Put the rest of the station into (X,Y,Z)
  DO 35 I = 1,NSEC
    X(I) = XWNG(I)
    Y(I) = YWNG(I)
    Z(I) = ZWNG(I)
  35 CONTINUE
  C"
**UNIX FORTRAN Program**

```
include "sgrid.com"

DIMENSION D1(NP1), D2(NP1), S(NP1)

Ktip = (Kdim+1)/2

C From the nose to the leading edge
C E= from station 1 to M1
C First point of the nose
DO 12 K=1,Kdim
  X(E1) = XIM(E1)
  Y(E1) = YIM(E1)
  Z(E1) = ZIM(E1)
12  CONTINUE

C Loop for all stations, from station 1 to station M1
DO 50 K=M2,M1
  L = K
  Xlocal = XIM(M)
  STORE THE INPUT TO DUMMY ARRAY
  DO 31 J=1,NP1
    YINT(J) = YIM(M,E)
    ZINT(J) = ZIM(M,E)
 31  CONTINUE
C TREP is the value of the first point of the wing
C TREP is the corresponding index of each station.
TREP(TOUT(1,NP1))

C OPTIONS -
C Boeing Baseline Configuration
DO 43 J=1,NP
  IF(YINT(J) >= YREF) THEN
    IREF = 4
    GOTO 44
  ENDIF
43  CONTINUE
C KEEP=NP/2 + KREFADD
C Langley's Low Room Configuration
C IREF = 31

C Lower part of the nose (-Y to YREF)
E=1
EE=KREF
E1 = EE - ES = 1
DO 74 J=1,KE
  E = E - ES = 1
  D(J,E1) = YINT(J)
  D2(J,E1) = ZINT(J)
74  CONTINUE
C CALL DISTARC(D1,D2,EX,THEN,IDIST,ETIP,PMBOT,1)
C DO 85 K=1,ETIP
  Y(L,K) = THEN(K)
  Z(L,K) = IDIST(K)
  X(L,K) = XLOCAL
85  CONTINUE
C From YREF to pos Y
E=KREF
EE=KREF
E1 = EE - ES = 1
DO 100 K=1,K,E
  E = E - ES = 1
  D(J,E1) = YINT(J)
  D2(J,E1) = ZINT(J)
100  CONTINUE
C CALL DISTARC(D1,D2,EX,THEN,IDIST,ETIP,PMBOT,0)
C DO 400 K=ETIP,KDIM
  Y(L,K) = THEN(K-ETIP+1)
  Z(L,K) = IDIST(K-ETIP+1)
  X(L,K) = XLOCAL
400  CONTINUE
C RETURN
```

**samgrid.f**
SUBROUTINE RESIST(XLOCST, ETIP, FAC, IFLAT)

include "agrid.com"

This is the main subroutine to redistribute the points from spanwise cut to streamwise cut.

When there is "cheap-SIMP", it means the output is for SIMP code

PARAMETER(NINT=400)

DIMENSION S(NINT)

NE = number of point in the upper surface
NL = number of point in the lower surface
NC = number of spanwise sections

XFLAT = the leading edge

IFLAT = 2 point of streamwise cut for =XLOCST at each surface

ET = # of pts in circuim. direction extracted from the old grid

EXIT = # of pts in the circuim. direction in one surface

IFLAG = 1: upper surface
IFLAG = 2: lower surface

IF(INT.LE.ETIP) THEN
  WRITE(*,*) 'SUB RESIST : INT is less than ETIP'
  STOP
ENDIF

IF(IFLAG.EQ.2) THEN
  NUL = NU
ELSE
  NUL = NL
ENDIF

The streamwise distance passes the leading edge

IF(XFLAT.GT.XE(DC)) THEN
  YSTATION = is at the wing tip
  IF(XE(DC).LE.XE(NC)+CHORD(NC)) THEN
    Should call WINGWAKE if wake is contained in this station
    IF(XE(DC).GT.XE(NC)+CHORD(NC)) THEN
      CALL WINGWAKE(XE(DC), ET, NUL, IFLAT)
      GOTO 200
    ENDIF
  ENDIF

Do 60 M1=NC
Do 50 M2=NU
  IF(XBASE(J,IFLAT,M-1).EQ.XLOCST.OR.
    ABS(XBASE(J,IFLAT,M)).LE.1.E-7) THEN
    X1 = XBASE(J,IFLAT,M-1)
    X2 = XBASE(J,IFLAT,M+1)
    Y1 = YBASE(J,IFLAT,M-1)
    Y2 = YBASE(J,IFLAT,M+1)
    Z1 = ZBASE(J,IFLAT,M-1)
    Z2 = ZBASE(J,IFLAT,M+1)
    CALL LININT(X1,X2,Y1,Y2,XLOCAL,YY)
    CALL LININT(X1,X2,Z1,Z2,XLOCAL,ZZ)
    YINT(JT) = YY
    ZINT(JT) = ZZ
    Go to 200
  ELSEIF (ABS(X1*X2).LE.1.E-7) THEN
    IF(ABS(X1).LE.1.E-7) THEN
      ET = 0
      Do 160 J = 1, NUL
      ET = ET + 1
      Create a point at the root
      IF(XBASE(J,IFLAT).GT.XLOCAL) THEN
        X1 = XBASE(J,IFLAT,1)
        X2 = XBASE(J,IFLAT,1)
        Y1 = YBASE(J,IFLAT,1)
        Y2 = YBASE(J,IFLAT,1)
        Z1 = ZBASE(J,IFLAT,1)
        Z2 = ZBASE(J,IFLAT,1)
        CALL LININT(X1,X2,Y1,Y2,XLOCAL,YY)
        CALL LININT(X1,X2,Z1,Z2,XLOCAL,ZZ)
        YINT(JT) = YY
        ZINT(JT) = ZZ
        Go to 200
      ENDIF
      Do 150 M = 2,NC
      X1 = XLOCST-XBASE(J,IFLAT,M-1)
      X2 = XLOCST-XBASE(J,IFLAT,M)
      Y1 = YLOCST-YBASE(J,IFLAT,M-1)
      Y2 = YLOCST-YBASE(J,IFLAT,M)
      IF(XFLAT.LT.LT.0.) THEN
        X1 = XBASE(J,IFLAT,M-1)
        X2 = XBASE(J,IFLAT,M)
        Y1 = YBASE(J,IFLAT,M-1)
        Y2 = YBASE(J,IFLAT,M)
        CALL LININT(X1,X2,Y1,Y2,XLOCAL,YY)
        CALL LININT(X1,X2,Z1,Z2,XLOCAL,ZZ)
        YINT(JT) = YY
        ZINT(JT) = ZZ
        Go to 160
      ELSEIF (ABS(X1*X2).LE.1.E-7) THEN
        IF(XBASE(J,IFLAT).LT.1.E-7) THEN
          IF(IFLAG(JFLAT).LT.1.E-7) THEN
            Go to 160
          ELSE
            Go to 160
          ENDIF
YINT(KT) = YBASE(1,1FLAT,M-1)
ELSE
YINT(KT) = YBASE(1,1FLAT,M)
ENDIF
ENDF
GOTO 160
ENDF
CONTINUE
CONTINUE
ENDIF
DO 220 EX=1,EX
Z(EE) = YINT(EE)
DO 230 EE=1,EX
YINT(EE) = S(EE-EX)
DO 240 EE=1,EX
S(EE) = ZINT(EE)
DO 250 EE=1,EX
ZINT(EE) = S(EE-EX)
CONTINUE
CONTINUE
CONTINUE
CONTINUE
C OPTIONS :
C Distribute C-ordinates (warp around direction) from root to
C leading edge and back (dist).
C IT=1, grid points will cluster near the wing tip, =0 near the root.
C For NSEC, IT=0 / For Boeing, IT=1
C CALL DISTARC(YINT, YINT, KT, ZDIST, YNEW, KT_P, FA, T)
C IF(L.NE.NSEC) CALL CSPLINE(ZINT, YINT, KT, ZDIST, YNEW, ETIP)
RETURN
END
SUBROUTINE SUBTRACGRID(NPL1,X,Y,Z,NSEC,KDIM)

This subroutine allows us to subtract a grid line NPL1 from a streamwise section, and the new dimension is NSec again.

DIMENSION X(NPL1),Y(NPL1,NPI),Z(NPL1,NPI)

REnumber the late stations
NSec = NSec-1
DO 10 L=NPL1,NSec
  X(L) = X(L+1)
  Y(L) = Y(L+1,1)
  Z(L) = Z(L+1,1)
10  CONTINUE
DO 30 K=1,KDIM
  CONTINUE
30  CONTINUE
RETURN
END
SUBROUTINE TAIL(FAC1,FAC2)

DIMENSION S(NPI)
COMMON /REF/ IROOT,KTIP,ABCOER

ETIP = (EDIM+1)/2
LE = L-1

PT1 = (TOUT(NSEC,1)+YOUT(NSEC,NPTS))/2.
PT2 = TIN(NF,NFP)+TIN(NF,1))/2.
X1 = X(LK)
X2 = X(IN(NF))

LI = L-1
L2 = L-1+(NF-M2)-1
DO 380 L=LI,L2 
M=M2-L-1-1
(I) = IZIN(M)
LOCAL = IZIN(M)

C OPTIONS -------------------------------

C WAKEPT is the Y value that the wake is (le Y value of IROOT)

C Tape of the configuration

C

C Design the old point (EF) will be the off fuselage side
C It is also the number of pts in the upper or lower fuselage, therefore
C it depends on the previous station.
C (no. of pts in this station)/ (no. of pts in previous) =
C (radius of this station) / (radius of previous)
C RATIO=ABS(TIN(N,1)-TIN(N,NFP))/ABS(TIN(N,1)-TIN(N-1,NFP))
C EF = EDIM-NPTS/2
C EF = EDIM-NPTS/2
C IF(RATIO<0.5 OR RATIO>2.1) THEN
C
C ENDIF

C Find ZTI the point at old grid where TIN(N,PTI)=WAKEPT
C and calculate the points from seg Y to Y at EF
C
DO 222 EF=1,NFP
C IF(RATIO(TIN(N,F)-WAKEPT),LE.1.E-7) THEN
EF=K
GOTO 223
C
ENDIF
C IF(RATIO(TIN(N,F),GF.WAKEPT)) THEN
Y=EF
C
ENDIF
C
EF=K
Y1=TIN(N,F,1)
Y2=TIN(N,F,K)

Y3=SIN(N,F,K)
C
Y=WAKEPT
C CALL LININT(Y1,Y2,Z1,Z2,YY)
C
YIN(N,F)=YY
C
GOTO 223
C
224 ENDIF

222 CONTINUE

223 CONTINUE

DO 230 E=1,EF
Y=EF
C
YIN(E)=YNEW
Z=LE*EDIST(E)
230 CONTINUE

C Find ZTI the point at old grid where TIN(N,PTI)=WAKEPT

C Note: EF=NFPT-PTI+1
C
C and calculate the points from seg Y to Y at EF
C
EF=NFPT-PTI+1

EF=K
Y=EF
C
EF=K
Y1=TIN(N,F,K-1)
Y2=TIN(N,F,K)

Y3=SIN(N,F,K-1)
C
EF=K
Y=WAKEPT
C CALL LININT(Y1,Y2,Z1,Z2,YY)
C
EF=K
YIN(E)=YNEW
Z=LE*EDIST(E)
240 CONTINUE

250 CONTINUE

C These are points in fuselage side
C E=No. of pts in the off-fuselage side
C EN=EF-M2+1
C RSPAN= ABS((S(LE,EF)-S(L,EF))
C IF (FAC2. LT. K,FAC2-1.1.E-5
C DELT=FAC2**2/RSPAN/FLOAT(KNF)
C CALL DISTY(DISTY,KNF+1,5.0)
C
C CALL LININT(X1,X2,YOUT(NSEC,NPTS)/2),PTZ,LOCAL,TOB
C
C DO 270 K=1,EN
C 2(L,EF-1)=Z(L,EF-1)+RSPAN
C CALL LININT(ZROOT,RSAN,WAKEPT,TOB,2(L,EF-1),YY)
C YIN(K)=-YY
C 270 CONTINUE

C DO 280 E=1,EN
C 2(L,EDIM-EF-1)=-Z(L,EF-E)
C YIN(K)=-YY
C 280 CONTINUE

C Smooth the fuselage-wake part
C
C DO 285 KE=1,EDIM
C
Y=KE-1
Z=KE+1
YIN(KE)=Y(L,KE)
Z=KE+1
Z=KE-1
285 CONTINUE

ZROOT = IN(N,F,TI)
RFILT = RFILT

IF( (L) GE 100 START AND .(L) LE 1000)
290 CONTINUE

C CALL FILE( YINT, ZINT, EDIM, MD1, MD2, MT1, MT2, RFILT)
DO 290 I=1,EDIM
   Y(I,E)=Y(TINT(E))
   I=1,E=I+1
   CONTINUE

DO 290 E=1,NPH
   VINT(E)=V(INM(E))
   CONTINUE

CALL DISTANCE(YINT, ZINT, NPH, THEM, EDIST, RTIP, 0.05, 1)

DO 295 E=1,NPH
   Z(E)-DIST(E)
   CONTINUE

DO 295 E=1,NPH
   V(E)=THEM(E)
   CONTINUE

CALL DISTANCE(YINT, ZINT, NPH, THEM, EDIST, RTIP, 0.05, 1)

DO 296 E=1,NPH
   ZINT(E)=ZINT(E+1)
   CONTINUE

DO 296 E=1,NPH
   VINT(E)=VINT(E+1)
   CONTINUE

DO 300 I=1,EDIM
   DO 313 E=1,NPH
      Z(E)=Z(E-1)
   END
   V(E)=V(E-1)
   CONTINUE

RETURN
END
SUBROUTINE WGRID

include "aerodynamics"

This subroutine reads the fuselage grid and combines it with the
wing grid.

DIMENSION D1(NP1),D2(NP1),L(NP1)

COMMON /REDF,ROOT,STIP,ARCOR

Read surfgrid dimensions

XIN,TIN,ZIN = read in grid
X1,2 = output grid
X3,XF,XZ,XQ = dummy vectors
EDIM = # of points of the whole body in the warp-around direction
EDIM1 = EDIM/2 + 1, EDIM2 = EDIM/2 + 2
CSW = 1 if nose bottom part (circumferential direction)
CSW2 = 0 if nose top part (circumferential direction)
CSW2 = 1 if tail part (circumferential direction)
RFIL = fillet radius
ARCOR = this makes fillet looks smooth
BLAKE = Distance of the trailing edge in body length
BL = body length
NUMW = Number points along the trailing edge
SCALE = Scale factor for the wing-body
NAMELIST /FOIL/ EDIM,EDIM1,EDIM2,FILT,ARCOR

Write(*,'(I2)',EDIM=0,EDIM1=0,EDIM2=0,FILT=0,ARCOR=0)

Read the fuselage geometry

CALL FUSE

Take this initial grid
1) convert it into area distribution
2) add shape functions
3) convert it back to grid

CALL FUSE

BL = XIN(NP1)
ECU = ECUE+BL
ETIP = (EDIM1+1)/2

Find number of streamwise stations. We keep the stations
of the fuselage. As to the common area of the wing and fuselage
we use the stations of the wing.
ML is the station the nose ends
MT is the station the nose ends
ML0 = 0
MT0 = 0
DO 10 M = 1,NP1 IF(Abs(XIN(M)-XOUT(1,1))) LE.1.E-7) ML=M
IF(Abs(XIN(M)-XOUT(NSEC,1))) LE.1.E-7) MT=M+1
10 CONTINUE
IF(M1.EQ.0) THEN DO 20 M=1,NP1 IF(XIN(M).LE.XOUT(1,1).AND.XIN(M).GT.XOUT(1,1))
20 CONTINUE
ENDIF
IF(M2.EQ.0) THEN DO 30 M=1,NP1 IF(XIN(M).LE.XOUT(NSEC,1).AND.XIN(M).GT.XOUT(NSEC,1))
30 CONTINUE
ENDIF

We divide the configuration into four parts:
the nose, the wing-body, the tail, and the extended wake. And
for each part we distribute points for the lower and upper part
separately.

From the nose to the leading edge
CALL NOSE(FB,FT,FTOP)

From the leading edge to the trailing edge (the whole wing)
CALL WINGANCE

From the trailing edge to the end of tail
CALL TAIL(-1,FAC)

Redistribute each streamwise station, so that the grid
is closer near the wing tip.
IF(CUTTER,GT.0.) CALL EDFACE

Redistribute the streamwise stations, so that
it is equally spaced.
IF(CUTTER,GT.0.) CALL EDSPACE

Add the extended wake
L2 = L-1
XLAST = X(L2) = BL+NUMW
TOTAL = XLAST + X(L2)
We might like to scale the whole thing by a reference length

Boeing Baseline Configuration
Before write out the grid, this Boeing wing needed to be fixed

Read in new surfgrid in plot3d format

Write the wing-body grid (x,y,z) to (xout,yout,zout)

END
SUBROUTINE WMATCH

include "sgr.incl"

DIMENSION OL(NP1),D2(NP2)

IF(OL(I).GT.D2(LF)) THEN

OL(I) = D2(I)

END IF

RETURN
**SOURCE TEXT**

SUBROUTINE WING_BODY

include "samgrid.com"

DIMENSION DI(NP1), D2(NP1)

COMMON/REF/, ECHOT, TIP, ARCS

LB = M + 1
LE = M + NSEC + 1
DO 200 L = LB, LE
200 L = L - 1
START WITH SECOND WING STATION

ILOCAL = XOUT(L, 1)

Calculate the points (TINT, ZINT) on the fuselage at ILOCAL.

Note: assumed that in this section, each station has same number
of points in nose around direction.

DO 10 M = NFP
10 IF (XIN(M) .GE. ILOCAL) THEN

MW = M
GOTO 15
ENDIF

CONTINUE

DO 30 K = 1, NFP
30 XI = XIN(M)
X2 = XIN(M - 1)
Y1 = YIN(M)
Y2 = YIN(M - 1)
Z1 = ZIN(M)
Z2 = ZIN(M - 1)
CALL LININT(XI, X2, Y1, Y2, XI, YINT(K))
CALL LININT(XI, X2, Z1, Z2, XI, ZINT(K))
YINT(K) = YINT(K) + 1
ZINT(K) = ZINT(K) + 1
CONTINUE

Desde la sección del ala, se tiene la sección de fuselaje.

LW es la sección index for YIN, XIN

EI is the last pt. / the fuselage & wing at bottom in old grid
EI is the last pt. / the fuselage & wing at top in old grid
MID is the last point./ the fuselage & wing at bottom in new grid

There are NPTS points in the wing area, used to check out
all wake points in the fuselage section.

EI is the last pt. / the fuselage & wing at bottom in old grid
EI is the last pt. / the fuselage & wing at top in old grid

MID is the last point./ the fuselage & wing at bottom in new grid

These are points from 1 to EI

EI - (NFP + 1) / 2
MID = (EI - NPTS) / 2

IF (YINT(K) .LT. YOUT(LW, 1)) .AND. YINT(K) .NE. YINT(K - 1) THEN

K1 = K
GOTO 72
ENDIF

DO 70 K = 1, NFP
70 IF (YINT(K) .LT. YOUT(LW, 1)) .AND. YINT(K) .NE. YINT(K - 1) THEN

K2 = K
GOTO 72
ENDIF

Do 70 K = 1, NFP
70 I = YINT(K)
J = YINT(K + 1)
K = YINT(K - 1)
ZINT(K) = ZINT(K) + 1
YINT(K) = YINT(K) + 1
CONTINUE

Calculate the points at the bottom (from seg. Y to Y at MID)

ES = XI
EE = XI
EN - EE = ES + 1

DO 82 K = ES + 1, EE
82 D1(K) = YINT(K)
D2(K) = ZINT(K)

CALL DISTARC(D1, D2, XNEW, ZDIST, MID, -10., 1)

DO 120 K = 1, MID
120 Y(L, K + MID + NPTS) = YNEW(K)
X(L, K + MID + NPTS) = XLOCAL
CONTINUE

For arrow-wing type, make sure wake points ok

IF (ABS(XOUT(LW, 1) - XOUT(LW, NPTS)) .GE. 1.E-6) THEN

Y(L, MID + NPTS) = YOUT(LW, 1)
Z(L, MID + NPTS) = ZOUT(LW, 1)
Y(L, MID) = Y(L, MID + NPTS + 1)
Y(L, MID) = Y(L, MID + NPTS + 1)

C For arrow-wing type, make sure wake points ok

IF (ABS(XOUT(LW, 1) - XOUT(LW, NPTS)) .GE. 1.E-6) THEN

Y(L, MID + NPTS) = YOUT(LW, 1)
Z(L, MID + NPTS) = ZOUT(LW, 1)
Y(L, MID) = Y(L, MID + NPTS + 1)
Y(L, MID) = Y(L, MID + NPTS + 1)
2252  \( Z(L,MID) = Z(L,MID+NPTS+1) \)

2254  ENDIF

2256  C

2257  C Fill the unsmooth part by FILET

2259  C First of all, find the set of points needed to be rearrange

2262  DO 125 KE=1, EDIM

2264  YINT(KE)=Y(L,KE)

2266  ZINT(KE)=Z(L,KE)

2269  125 CONTINUE

2272  RFILT = RFILT

2275  IF(X(L).GE.XSTART .AND. X(L).LE.XOFF)

2277  & CALL FILET(YINT, ZINT, EDIM, MB1, MB2, MT1, MT2, RFILT)

2280  DO 140 K=1, EDIM

2282  Y(L,K)=YINT(K)

2284  Z(L,K)=ZINT(K)

2287  140 CONTINUE

2290  200 CONTINUE

2293  RETURN

2296  END
SUBROUTINE WINGGRID(XLOCAL,ET,NUL,IFLAT)
  include "sgrid.com"

  This subroutine creates the data when the station is in the plane where some points are on the wing, some are the wake.

  At the x-sweeping, we put 10 points in the wake part.

  ET = 0.

  IF(ASSERT.GT.0.) THEN
    The wing is swept backwards
    DO 75 J=2,NC
      J1=J-1
      J2=J+1
      IF(XLOCAL.GT.XBASE(I_IFLAT,J)) GOTO 75
      XI1=XBASE(I_IFLAT,J)+CHORD(J)
      Z1=BASE(NUL,IIFLAT,J1)
      XI2=XBASE(I_IFLAT,J)+CHORD(J)
      Z2=BASE(NUL,IIFLAT,J2)
      Y1=FLOAT(M)*DZ
      Y2=FLOAT(M-1)*DZ
      CALL LININT(XI1,XI2,Y1,Y2,Z1,Z2,ZBASE(NUL,IIFLAT,J),ZROOT)
      CALL LININT(X1,X2,Y1,Y2,ZTIP,ZBASE(NUL,IIFLAT,J),YY)
      ZINT=M-1
      ZROOT=ZBASE(NUL,IFLAT,J)
      IF(NLEDG.EQ.MC) THEN
        NMC=NMC-M+1
        ELSE
      ENDIF
      IF(NMC=NC) THEN
        RT = (NMC-MC-1)+1
        ELSE
      ENDIF
    ENDIF
    CONTINUE
  ENDIF
  CONTINUE

  END

  DO 65 M=NLEDG,2,-1
    DO 64 J=2,NUL
      IF(XLOCAL.GT.BASE(J_IFLAT,M)) THEN
        X1 = BASE(J_IFLAT,M-1)
        X2 = BASE(J_IFLAT,M)
        Y1 = YBASE(J_IFLAT,M-1)
        Y2 = YBASE(J_IFLAT,M)
        Z1 = BASE(J_IFLAT,M-1)
        Z2 = BASE(J_IFLAT,M)
        CALL LININT(X1,X2,Y1,Y2,Z1,Z2,ZBASE(J_IFLAT,M))
        CALL LININT(X1,X2,Y1,Y2,ZTIP,ZBASE(J_IFLAT,M),YY)
        ZINT=F+1
        ZROOT=ZBASE(J_IFLAT,MC)
        IF(NLEDG.EQ.MC) THEN
          NMC=NMC-M+1
          ELSE
        ENDIF
        IF(NMC=NC) THEN
          RT = (NMC-MC-1)+1
          ELSE
        ENDIF
      ENDIF
    ENDIF
    CONTINUE
  ENDIF

  Add 10 points for the wake.
  ZROOT = BASE(NUL,IIFLAT,1)
  Y1=YBASE(NUL,IIFLAT,1)
  Y2=YBASE(NUL,IIFLAT,2)
  Z1=BASE(NUL,IIFLAT,1)
  Z2=BASE(NUL,IIFLAT,2)
  CALL LININT(X1,X2,Y1,Y2,Z1,Z2,ZROOT)
  CALL LININT(X1,X2,Y1,Y2,ZTIP,ZROOT,YY)
  ZINT=0
  ZROOT=ZBASE(NUL,IIFLAT,1)
  IF(NLEDG.EQ.MC) THEN
    NMC=NMC-M+1
    ELSE
  ENDIF
  IF(NMC=NC) THEN
    RT = (NMC-MC+1)+1
    ELSE
  ENDIF
  GO TO 68
  CONTINUE

  END
TINT(M) = YYY
DO 100 M=1,1,-1
   DO 90 I=2,NUL
      IF(LOCAL.LE.XBASE(I,IFLAT,M)) THEN
         XI = XBASE(I-1,IFLAT,M)
         X2 = XBASE(I,IFLAT,M)
         Y1 = YBASE(I-1,IFLAT,M)
         Y2 = YBASE(I,IFLAT,M)
         Z1 = ZBASE(I-1,IFLAT,M)
         Z2 = ZBASE(I,IFLAT,M)
      CALL LININT(X1,X2,Y1,Y2,XLOCAL,YY)
      CALL LININT(X1,X2,Z1,Z2,XLOCAL,ZZ)
      YINT(J1-N+I+10) = YY
      ZINT(J1-N+I+10) = ZZ
   90    CONTINUE
100   CONTINUE
IF = IF + 10 + J1
GO TO 200
CONTINUE
END
C SUBROUTINE READIN
C
C include "sgrid.com"
C
NC = # of sections in the spanwise direction
XBASE(i) = X value of i-th section
YBASE(i) = leading edge X value of i-th section
YCHORD(i) = Chord length of the i-th section

Note: the end points of upper and lower sections are same physical pts.

C MACHE configuration
NL = NU
K = 1
CONTINUE
READ(10,900) NC,NU
READ(10,920) XBASE(1,1),YBASE(1,1),YCHORD(1)
READ(10,940) XBASE(1,2)
DO 15 I = 1,NU
READ(10,940) XBASE(1,1),YBASE(1,1),YCHORD(1)
XBASE(I,1) = XBASE(1,1)
YBASE(I,1) = YBASE(1,1)
CHORD(I) = ABS(XBASE(I,1)-XBASE(NU,1))
IF(K .GT. NU) GOTO 11
IF(K = NU) GOTO 11
C
C 900 FORMAT(12)
C 920 FORMAT(25E15.15,9X,15/)
C 930 FORMAT(12)
C 940 FORMAT(12)
C 950 FORMAT(12)
C 960 FORMAT(12)
C
RETURN
END
SUBROUTINE FUSEIN

include 'sgrid.com'

DIMENSION YT(NPI),ZT(NPI)

NF = # of section in the fuselage
NFP = # of points in nth section

Read the fuselage geometry

MACES configuration
READ(10,810)
READ(10,820)
READ(10,830)NF,NFP

M = 0

DO 100 M = 1,NF

CALL DRI Boundary(YT,E,NFP,YT,E,NFP,-10.,0)

DO 110 M = 1,NF

YT(M) = YT(E)
ZT(M) = ZT(E)

110 CONTINUE

IF(N.EQ.NF) GOTO 40

Write the fuselage geometry into PLOT3D Planar format.

EX = 1

WRITE(5)EX,EX,N

DO 800 EX = 1,N

WRITE(5)EX,EX,N

WRITE(5)ZT(1),ZT(1),EX

WRITE(5)ZT(1),ZT(1),EX

WRITE(5)YT(1),YT(1),EX

WRITE(5)YT(1),YT(1),EX

WRITE(5)EX,EX,N

800 CONTINUE

RETURN

END
SUBROUTINE MACIN
include "grid.com"
DIMENSION IT(MPI),ET(MPI)
COMMON /IAC/ XAC(MPI),YAC(MPI),ZAC(MPI),MAGS(MPI)
COMMON /HDIM/ MACH,MAACP,INDM(4)

C MAC = # of section in the array
C MAACP = # of points in Mth section
C
C Read the section geometry
READ(10,810)
READ(10,820)
READ(10,830),MACH,MAACP
N=0
40 CONTINUE
C
READ(10,840)
M = M + 1
C
DO 100 I=1,MAACP
READ(10,850),XAC(M),YAC(M),ZAC(M)
C
100 CONTINUE
C CALL DISTANCY(TT,ET,MAACP,TT,ET,MAACP,-18,0)
C
DO 110 I=1,MAACP

C Write the section geometry into PILOT3D FLeX3 format
C
X=1
Y=MACH!
WRITE(32,111),X,M
DO 120 I=1,M

C
WRITE(32,112),XAC(I),YAC(I),ZAC(I)

110 CONTINUE
C
C CALL flush (32)
C
END
C
C
RETURN
END
SUBROUTINE WINGMAKER

C include 'sgrid.com'
C
C This program generates a 'clipped' delta wing with no twist
C based on airfoil coordinates read in from fort.90. To use as
C part of sgrid, WINGMA is not necessary (nor is VARISNEE).
C Written by Donovan L. Mathias
C July 1982
C
C Whenever possible the same variables are used as in sgrid.f
C Fort.90 airfoil coordinates
C
C Description of the wing
C
C Declarations
C
REAL XTE(150),SLOPE1,SLOPE2,SCALE,YAF(150)
REAL YU(150),YT(150)
REAL SPAN
INTEGER L,1,1,E

C Initialization
C
read(77,*),NC
read(77,*,XLE(1),XLE(NC)
read(77,*,XTE(1),XTE(NC)
read(77,*,ZBASE(1,1,1),ZBASE(1,1,NC)

C Need airfoil coordinates (L is # of A coords.)
C
READ(90,*),
READ(90,*)H
READ(90,*)N
H=N
DO I=1,N,
READ(90,19),XAF(I),YT(I),YAF(I)
ENDDO

C Establish lift distance (Spanwise)
C
format(3X,9,3X,9,3X,9,3X,9,3X,9,3X,9,3X,9,3X,9)

C SPAN = ZBASE(1,1,NC) - ZBASE(1,1,1)
C
DO X=9,NC-1
DO I=1,N,
ZBASE(1,1,E+1) = (X*SPAN/(NC-1))
ZBASE(1,2,E+1) = (X*SPAN/(NC-1))
ENDDO

C Establish sweep (1 FOR LE, 2 FOR TE)
C
SLOPE1 = (XLE(NC)-XLE(1))/(ZBASE(1,1,NC)-ZBASE(1,1,1))
SLOPE2 = (XTE(NC)-XTE(1))/(ZBASE(1,1,NC)-ZBASE(1,1,1))

C Generate leading and trailing edges
C
DO X=1,NC
XLE(X) = XLE(1) + SLOPE1*ZBASE(1,1,1)
XTE(X) = XTE(1)+SLOPE2*ZBASE(1,1,1)
ENDDO

C Distribute grid points
C
DO X=1,NC
SCALE = XAF(X)/XLE(X)
DO I=1,N,
ZBASE(1,1,E) = XBASE(1,1,E) + SCALE*XAF(I)
ZBASE(1,2,E) = YU(I)*SCALE
ZBASE(1,3,E) = YT(I)*SCALE
ENDDO

C Return values to original names
C
DO X=1,NC
CHORD(X) = ABS(XLE(E)-XTE(E))
YAF(X) = YBASE(1,1,E)
ENDDO
RETURN
END
Manual of PVM

Samson Cheung

Merritt Smith

Local console

Ethernet
# Table of Contents

1 Preface ........................................................................ 1

2 Introduction
   Software Package .................................................. 2
   Definitions ................................................................. 2
   Structure of PVM ...................................................... 3
   Directory Setup .......................................................... 3

3 Programming Concepts
   Master and Slaves .................................................. 4
   Single Program-Multiple Data (SPMD) ......................... 6

4 PVM Daemon
   Console Commands .................................................. 10
   Console Usage ........................................................ 11

5 PVM Library
   Process Control ..................................................... 12
   Dynamic Configuration .............................................. 13
   Message Buffers .................................................... 13
   Packing and Unpacking ............................................. 13
   Sending and Receiving ............................................. 14

6 Tutorial
   Golden Section ...................................................... 16
   Serial Program ....................................................... 17
   PVM Master Guideline/Master Program ....................... 18
   PVM Slave Guideline/Slave Program ......................... 22
   Compilation and Running ........................................ 24
Preface

This manual serves as a supplementary document for the official reference manual of a relatively new research software, PVM, which has been developed at Oak Ridge National Laboratory. A beginner, who has no previous experience with PVM, would find this manual useful.

We would like to thank you in advance that if you find any problems in PVM or this manual, please contact one of us.

Mr. Merritt Smith  
NASA Ames Research Center, MS 258-1  
Moffett Field, CA 94035  
e-mail: mhsmith@nas.nasa.gov  
phone: (415) 604-4462

Dr. Samson Cheung  
MCAT Institute  
NASA Ames Research Center, MS 258-1  
Moffett Field, CA 94035  
e-mail: cheung@nas.nasa.gov  
phone: (415) 604-4462
This manual provides you with an introduction to PVM and provides the fundamentals necessary to write FORTRAN programs in the PVM environment through a tutorial sample. This manual is designed for those who have no previous experience with PVM. However, you should know basic FORTRAN programming and UNIX. If you are ready for an advanced PVM application, please consult the official PVM Reference Manual.

PVM stands for Parallel Virtual Machine. It is a software package that allows a heterogeneous network of parallel and serial computers to appear as a single concurrent computational resource. PVM allows you to link up all or some of the computational systems on which you have accounts, to work as a single distributed-memory (parallel) machine. We call this a Virtual Machine.

PVM is useful for the following reasons. Unlike large mainframe computers or vector supercomputers, workstations spend most of the time idle. The idle time on a workstation represents a significant computational resource. PVM links these workstations up to become a powerful multi-processor computational machine. With PVM, the lack of supercomputer resources should not be an obstacle to number crunching computational programs. Furthermore, the annual maintenance costs of a vector supercomputer is often sufficient to purchase the equivalent computing resource in the form of workstation CPU's.

Here are some terms we use throughout this document:

**Virtual Machine** PVM links different user-defined computers together to perform as one large distributed-memory computer. We call this computer the Virtual Machine.

**Host** Individual computer (member) in the virtual machine.

**Process** Individual program operating on different computers or hosts.

**Processor** The processing unit in computers. A virtual machine can be viewed as a multi-processor computer.
INTRODUCTION

Task
The unit of computation handled by the virtual machine. You may want to think of one processor handling one task.

Tid
Task identification number which is a unique number used by the daemon and other tasks.

Console
A program from which you can directly interact with the virtual machine. (Add hosts, kill processes,...)

Structure of PVM
The PVM software is composed of two parts. The first part is a daemon. We call it *pvmd*. This is the control center of the virtual machine. It is responsible for starting processes, establishing links between processes, passing messages, and many other activities in PVM. Since the daemon runs in the background, you have to use PVM console to directly interact with the virtual machine.

The second part of the system is a library of PVM interface routines located in *libpvm3.a*. This library contains user callable routines for message passing, spawning processes, coordinating tasks, and modifying your machine. In writing your application, you will need to call the routines in this library.

Directory Setup
This setup is for NAS system. Before you use PVM, you need to set up the following directories on all the machines that you want PVM to link:

- Make a directory `$/iol/pvm3/bin/ARCH` in all the hosts of the virtual machine.

  **Note** `ARCH` is used throughout this manual to represent the architecture name that PVM uses for a given computer. The table in the Appendix lists all the ARCH names that PVM supports. For example, for Silicon Graphic IRIS workstations, you should make a directory `$/HOME/pvm3/bin/SGI`.

- Make a directory `$/HOME/pvm3/include`, and copy the file `fpvm3.h` from `/usr/nas/pkg/pvm3.2/include`. (If you are on different system from NAS, please consult your system consultant.)

- Make a directory `$/HOME/pvm3/codes`, and write your application programs in this directory. You can actually put your programs anywhere you like as long as the correct "include" files are included. The current setup is for clarity.
Unlike graphical software or a word-processor, you cannot see PVM working by clicking your mouse buttons. In fact, a virtual machine is quite an abstract concept because you don't physically have a multi-processor machine! In this chapter, you will learn a simple concept, which will help you to visualize how PVM works.

A common way to work with PVM is a Master/Slave relationship. A Master process starts Slave routines and distributes work. However, a Master does not actively participate in the computation. A Master process most often resides on the originating host (user's computer), while the Slave programs are distributed to the hosts of the virtual machine.

You need to distribute executables of Slave programs to the directory $HOME/pvm3/bin/ARCH on every host. You can locate this Master program anywhere you like.

Since the Master program spawns Slave programs on each of the hosts to do jobs, it is important to understand the communication (message passing) among the hosts in PVM.

Typically, a Master and a Slave have the following logic:

<table>
<thead>
<tr>
<th>Master</th>
<th>Slave</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Enroll itself to PVM</td>
<td>1 Enroll itself to PVM</td>
</tr>
<tr>
<td>2 Spawn slave processes</td>
<td>2 Receive message from master</td>
</tr>
<tr>
<td>3 Initialize buffer, pack, and send message to all slaves.</td>
<td>3 ...do something useful...</td>
</tr>
<tr>
<td>4 ...wait for slaves to finish...</td>
<td>4 Initialize buffer, pack, and send message to master</td>
</tr>
<tr>
<td>5 Receive message from slave(s)</td>
<td>5 Exit PVM</td>
</tr>
<tr>
<td>6 Exit PVM</td>
<td></td>
</tr>
</tbody>
</table>

The figure on the opposite page graphically describes a Master/Slave relationship and shows the exchange of information.
FIGURE 1. Communication in *Master/Slave* programs.
Another common way to work with PVM is the *SPMD*, Single Program Multiple Data model. There is only a single program, and there is no Master program directing the computation. The user starts the first copy of the program and using the routine `pvmfparent()`, this copy can determine that it was not spawned by PVM, and thus must be the first copy (parent). It then spawns multiple copies (children) of itself and passes them the array of *tids*. At this point each copy is equal and can work on its partition of the data in collaboration with the other processes.

Typically, a SPMD program has the following logic:

1. **Enroll in pvm**
2. **If I am the first copy (parent)**
   a) Spawn child processes
   b) Initialize buffer, pack, and send message out
3. **If I am a secondary copy (child)**
   Receive messages
4. **Work!...Work!...Work!**
5. **Exit PVM**

The program on the opposite page describes a SPMD logic and shows the exchange of information. Please spend some time to study the program.

In the next chapter we will introduce the PVM daemon and the fundamentals of message passing.
**SPMD Program**

```
c SPMD Fortran example using PVM 3.0

program spmd
include '../include/fpvm3.h'
PARAMETER( NPROC=4 )
integer mytid, me, i
tid(0:NPROC)

c Enroll in pvm
call pvmfmytid( mytid )

c Find out if I am parent or child
call pvmfparent( tid(0))
if( tid(0) .lt. 0 ) then
tid(0) = mytid
me = 0

c start up copies of myself
call pvmfspawn('spmd',PVMDEFAULT,'*',
* NPROC-1,tid(1), info)

c multicast tid array to children
call pvmfinitSend( PVMDEFAULT, info )
call pvmfpack( INTEGER4, tid, NPROC, i, info)
call pvmfmcast( NPROC-1, tid(1), 0, info )
else

c receive the tid array and set me
call pvmfrecv( tid(0), 0, info )
call pvmfunpack( INTEGER4, tid, NPROC, l, info)
do 30 i=l, NPROC-1
   if( mytid .eq. tid(i) ) me = i
30 continue
endif

c all NPROC tasks are equal now
c and can address each other by tid(0) thru tid(NPROC-1)
c for each process me => process number [0-(NPROC-1)]
print*, 'me = ', me, ' mytid = ', mytid
call dowork( me, tid, NPROC )

program finished exit pvm
call pvmexit(info)
stop
end
```
Notes
subroutine dowork( me, tids, nproc )
include '../include/fpvm3.h'

C Simple subroutine to pass a token around a ring
C ...........................................................

integer me, nproc
integer tids( 0:nproc)

integer token, dest, count, stride, msgtag

count = 1
stride = 1
msgtag = 4

if( me .eq. 0 ) then
  token = tids(0)
call pvmfinitsend( PVMDEFAULT, info )
call pvmfpack( INTEGER4,token,count,stride,info)
call pvmfsend( tids(me+1), msgtag, info )
call pvmfrecev( tids(nproc-1), msgtag, info )
print*, 'token ring done'
else
  call pvmfrecev( tids(me-1), msgtag, info )
call pvmfunpack( INTEGER4,token,count,stride,info)
call pvmfinitsend( PVMDEFAULT, info )
call pvmfpack( INTEGER4,token,count,stride,info)
dest = tids(me+1)
if( me .eq. nproc-1 ) dest = tids(0)
call pvmfsend( dest, msgtag, info )
endif

return
end
The PVM daemon is the control center of the virtual machine. You can activate the PVM daemon by starting the PVM console or by invoking the daemon directly with a list of hosts. The latter will be discussed in chapter 6. To start the console, enter `pvm` at UNIX prompt on your local machine. The PVM console prints the prompt

```
pvm>
```

and accepts commands from standard input. The console allows interactive adding and deleting of hosts to the virtual machine as well as interactive starting and killing of PVM processes. Even if the daemon is started directly, the console can be used to modify the virtual machine.

### Console Commands

Here are the commands available in the PVM console:

- **ADD**: add other computers (hosts) to PVM
- **ALIAS**: define and list command aliases you set
- **CONF**: show members in virtual machine
- **DELETE**: remove hosts from pvm
- **ECHO**: echo arguments
- **HALT**: stop all pvm processes and exit daemon
- **HELP**: print this information
- **ID**: print console task identity
- **JOBS**: display list of running jobs
- **KILL**: terminate tasks
- **MSTAT**: show status of hosts
- **PS**: list tasks
- **PSTAT**: show status of tasks
- **QUIT**: exit PVM console, but PVM daemon is still activated
- **RESET**: kill all tasks
- **SETENV**: display or set UNIX environment variables
- **SIG**: send signal to task
- **SPAWN**: spawn task
- **UNALIAS**: remove alias commands you previous set
- **VERSION**: show PVM version
Suppose the console is running on workstation \textit{win210}. This computer will automatically be a host in your virtual machine. Here are some examples of using PVM console:

1. **Activate PVM console**
   
   \texttt{win210> pvm}

2. **Add amelia and fred to your virtual machine**
   
   \texttt{pvm> add amelia}
   \begin{verbatim}
   1 successful
   HOST  DTID
    amelia  c0000
   \end{verbatim}
   
   \texttt{pvm> add fred}
   \begin{verbatim}
   1 successful
   HOST  DTID
     fred  100000
   \end{verbatim}

3. **Check the configuration of your virtual machine**
   
   \texttt{pvm> conf}
   \begin{verbatim}
   3 host, 1 data format
   HOST  DTID  ARCH  SPEED
    win210  40000  SGI  1000
    amelia  c0000  SGI  1000
     fred  100000  SGI  1000
   \end{verbatim}

4. **Delete amelia**
   
   \texttt{pvm> delete amelia}
   \begin{verbatim}
   1 successful
   HOST  STATUS
    amelia  deleted
   \end{verbatim}

5. **Exit PVM console, but PVM daemon is still running**
   
   \texttt{pvm> quit}
   \begin{verbatim}
   pvmd still running
   \end{verbatim}
   
   \texttt{win210>}

---
This chapter introduces the PVM library. In writing your application programs, you need to call the subroutines in the library to instruct PVM to control processes, send information, pack/unpack data, and send/receive messages. Many subroutines have pre-defined option values for some arguments. These are defined in the include file *epvm3.h* and are listed in the Appendix.

**Process Control**

**call pvmfmytid(tid)**

This routine enrolls this process with the PVM daemon on its first call, and generates a unique *tid*. You call this routine at the beginning of your program.

**call pvmfexit(info)**

This routine tells the local PVM daemon that this process is leaving PVM. You call this routine at the end of your program. Values of *info* less than zero indicate an error.

**call pvmfkill(tid,info)**

This routine kills a PVM task identified by *tid*. Values of *info* less than zero indicate an error.

**call pvmfspawn(pname,flag,where,ntask,tids,numt)**

This routine starts up *ntask* instances of a single process named *pname* on the virtual machine. Here are the definition of the other arguments:

<table>
<thead>
<tr>
<th>flag</th>
<th>Option Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PVMDEFAULT (0)</td>
<td>PVM can choose any machine to start task</td>
<td></td>
</tr>
<tr>
<td>PVMHOST (1)</td>
<td>where specifies a particular host</td>
<td></td>
</tr>
<tr>
<td>PVMARCH (2)</td>
<td>where specifies a type of architecture</td>
<td></td>
</tr>
<tr>
<td>PVMDEBUG (4)</td>
<td>start up processes under debugger</td>
<td></td>
</tr>
<tr>
<td>PVMTRACE (8)</td>
<td>processes will generate PVM trace data</td>
<td></td>
</tr>
</tbody>
</table>

*where* is where you want to start the PVM process. If *flag* is 0, *where* is ignored.

*tids* contains identification numbers of PVM processes started by this routine.

*numt* indicates how many processors started; negative values indicate an error.

**Note** You should always check *tids* and *numt* to make sure all processes started correctly.
call pvmfparent ( tld )
This routine returns the uid of the process that spawned this task. If the calling process was not created with pvmpspawn, then tld=PvraNoParent.

Dynamic Configuration

call pvmfaddhost( host, info )
call pvmfdelhost( host, info )
These routines add and delete hosts to the virtual machine respectively. Values of info less than zero indicate an error.

Note Both routines are expensive operations that require the synchronization of the virtual machine.

Message Buffers

call pvmfinitsend( encoding, bufid )
This routine clears the send buffer, and creates a new one for packing a new message.

<table>
<thead>
<tr>
<th>Encoding Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PVMDEFAULT (0)</td>
<td>XDR encoding if virtual machine configuration is heterogeneous</td>
</tr>
<tr>
<td>PVMRAW (1)</td>
<td>no encoding is done. Messages are sent in their original format.</td>
</tr>
<tr>
<td>PVMINPLACE (2)</td>
<td>data left in place. Buffer only contains sizes and pointers to the sent items.</td>
</tr>
</tbody>
</table>

bufid contains the message buffer identifier. Values less than zero indicate an error.

call pvmffreebuf( bufid, info)
This routine disposes the buffer with identifier bufid. You use it after a message has been sent, and is no longer needed. Values of info less than zero indicate an error.

Packing and Unpacking

call pvmfpack( what, xp, nitem, stride, info )
call pvmfunpack( what, xp, nitem, stride, info )
These routines pack/unpack your message xp, which can be a number or a string. You can call these routines multiple times to pack/unpack a single message. Thus a message can contain several arrays, each with a different data type.
PVM Library

Note There is no limit to the complexity of the packed messages, but you must unpack them exactly as they were packed.

Indicates what type of data xp is:

- STRING (0)
- REAL (4)
- BYTE1 (1)
- COMPLEX8 (5)
- INTEGER2 (2)
- REAL8 (6)
- INTEGER4 (3)
- COMPLEX16 (7)

nitem is the number of items in the pack/unpack. If xp is a vector of 5, nitem is 5.

stride is the stride to use when packing.

info is status code returned by this routine. Values less than zero indicate an error.

Sending and Receiving

call pvmfsend( tid, msgtag, info )
This routine labels the message with an integer identifier msgtag, and sends it immediately to the process tid. Values of info less than zero indicate an error.

call pvmfmcast( ntask, tids, msgtag, info )
This routine labels the message with an integer identifier msgtag, and broadcasts the message to all ntask number of tasks specified in the integer array tids. Values of info less than zero indicate an error.

call pvmfrecv( tid, msgtag, bufid )
This routine blocks the flow of your program until a message with label msgtag has arrived from tid. A value of -1 in msgtag or tid matches anything (wildcard). This routine creates a new active receive buffer, and puts the message in it. Values of bufid identify the newly created buffer; values less than zero indicate an error.

call pvmfrecv( tid, msgtag, bufid )
This routine performs in the same way as pvmfrecv, except that it does not block the flow of your program. If the requested message has not arrived, this routine returns bufid=0. This routine can be called multiple times for the same message to check if it has arrived, while performing useful work between calls. When no more useful work can be performed, the blocking receive pvmfrecv can be used for the same message.

call pvmfprobe( tid, msgtag, bufid )
This routine checks if a message has arrived; however, it does not receive the message. If the requested message has not arrived, this routine returns bufid=0. This routine can
be called multiple times for the same message to check if it has arrived, while performing useful work between calls.

call pvmtbufinfo (bufid, bytes, msgtag, tid, info)
This routine returns information about the message in the buffer identified by bufid. The information returned is the actual msgtag, source tid, and message length in bytes. Values of info less than zero indicate an error.
This chapter shows you how PVM may be applied to your application programs through a simple example. The example chosen is the Golden Section rule for finding the maximum of a function. You may remember it from Math class in high school. Let us review the method and the algorithm.

Suppose we want to find the maximum of a curve $y=f(x)$; where $x$ is between the interval $a_1$ and $a_2$. The points $a_3$ and $a_4$ are symmetrically placed in this interval, so that

$$a_3 = (1-\alpha) a_1 + \alpha a_2$$

(EQ 1)

$$a_4 = \alpha a_1 + (1-\alpha) a_2$$

(EQ 2)

See Figure 1 at left. Golden Section rule requires $\alpha$ to be 0.382.

The algorithm of finding the maximum is as follow:

<table>
<thead>
<tr>
<th>If $f(a_4) &lt; f(a_3)$</th>
<th>If $f(a_4) &gt; f(a_3)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Consider new interval $(a_1,a_4)$</td>
<td>1 Consider new interval $(a_3,a_2)$</td>
</tr>
<tr>
<td>2 Apply EQ. (1) and (2) again</td>
<td>2 Apply EQ. (1) and (2) again</td>
</tr>
<tr>
<td>3 Until maximum is reached</td>
<td>3 Until maximum is reached</td>
</tr>
</tbody>
</table>

If $f(a_3)=f(a_4)$, the maximum is found.

The FORTRAN program (Serial Program) on the opposite page is the Golden Section rule that a programmer would write on a normal serial computer. Please spend a few minutes to study the flow of the program. This simple program consists of two parts, the main (calling) program and the function subroutine. The latter has only four lines.

**Note** Notice that for each interval $(a_1,a_2)$, we need to call the function evaluation four times to find $f(a_1)$, $f(a_2)$, $f(a_3)$, and $f(a_4)$. 

Figure 1. Interval division for Golden Section
Serial Program

Linear optimization:
Search for maximum of a x-y curve.

DIMENSION A(4), FN(4)

Initial interval
L = 0
TOL = 1.E-3
A(1) = 0.4
A(2) = 1.6

ALPHA = 0.382

CONTINUE

Loop begins:
L = L + 1

Four function evaluations

A(3) = (1.-ALPHA)*A(1) + ALPHA*A(2)
A(4) = ALPHA*A(1) + (1.-ALPHA)*A(2)

FN(1) = F(A(1))
FN(2) = F(A(2))
FN(3) = F(A(3))
FN(4) = F(A(4))

WRITE(10,*) ' ', A(1), A(2), A(3), A(4)
WRITE(10,*) ' ', FN(1), FN(2), FN(3), FN(4)
WRITE(10,*) ' ', ERR = ABS(FN(2)-FN(3))
IF(ERR.LE.TOL) GOTO 999

IF(FN(4) .GT. FN(3)) THEN
B1 = A(3)
B2 = A(2)
A(1) = B1
A(2) = B2
GOTO 10
ELSEIF(FN(4) .LT. FN(3)) THEN
B1 = A(1)
B2 = A(4)
A(1) = B1
A(2) = B2
GOTO 10
ENDIF

FUNCTION F(X)
F = TANH(X)/(1.+X*X)
RETURN
END
PVM Master Guideline

Recall that in the procedure of finding a new interval, the program calls the function evaluation four times *serially* to get \( f(a_1), f(a_2), f(a_3), \) and \( f(a_4) \). We would like to assign four processors to perform the four function evaluations *simultaneously* on the virtual machine. Therefore, we modify the Serial Program by writing the main (calling) program as a Master program, and the function subroutine as a Slave program.

The following steps are general guidelines to writing a Master program. Please study the steps, and compare them with the program on the opposite page. Also compare it with the Serial Program.

1. **Include fpvm3.h**
   Include this file in your program, you are able to use the PVM preset variables; such as `PVMDFAULT`, `REAL4`, and more, mentioned in Chapter 4 and the Appendix.

2. **Enroll Master to PVM**
   Use `pvmfmytid(mytid)` to enroll.

3. **Assign virtual processors**
   Use the following call to spawn `nproc` function processes.
   ```c
   pvmfspawn (pname, PVMDFAULT, where, nproc, tids, numt)
   ```
   Also tell PVM the name of the Slave program (`pname`). PVM returns `tids`, the identifier of the `nproc` processors.

4. **Initialize buffer and pack data**
   Use `pvmfinitsend` to clear buffer.
   Use the following routine to pack a real array `A` of dimension `m`.
   ```c
   pvmfpack (REAL4, A, m, 1, info)
   ```

5. **Send message**
   Use the following call to send the packed message to the Slave process identified by `tids`.
   ```c
   pvfmcast(nproc, tids, msgtag, info)
   ```
Master Program

Linear optimization:
Search for maximum of a x-y curve.

PROGRAM MASTER

include '../include/fpvm3.h'
DIMENSION A(4),FN(4)
integer tids(0:32),who
character*8 where
character*12 pname

Enroll this program in PVM
call pvmfmytid(mytid)

Start up the four processors
nproc = 4
where = '*' 
pname = 'function'
call pvmfspawn(pname,PVMDFAULT,where,nproc,tids,numt)
do 20 i=0,nproc-1
   write(*,*) 'tid', i, tids(i)
20 continue

Initial interval
L = 0
A(1) = 0.4
A(2) = 1.6
ALPHA = 0.382
TOL = 1.E-3
ERR = 1.

CONTINUE

Loop begins:
L = L + 1
Equations (1) and (2)
A(3) = (1.-ALPHA)*A(1) + ALPHA*A(2)
A(4) = ALPHA*A(1) + (1.-ALPHA)*A(2)

Broadcast data to all node programs
first pack them, then send them
call pvmfinitsend(PVMDEFAULT,info)
call pvmfpack(INTEGER4,nproc,1,1,info)
call pvmfpack(INTEGER4,tids,nproc,1,info)
call pvmfpack(REAL4,A,4,1,info)
call pvmfpack(REAL4,ERR,1,1,info)

Pack nproc, tids, A, and ERR

msgtype = 1
call pvmfmcast(nproc,tids,msgtype,info)
6. **Wait until messages come from Slaves**
   Use `pvmrecv()` to block until Slaves return function values.
   Make sure value of `msgtype` matches values coming from Slaves.

7. **Receive and Unpack data**
   The sequence of unpacking is the same as the packing in the Slave.

8. **Exit PVM**
   Use `pvmexit(info)` to exit PVM.
Wait for results from processors

msgtype value matches the one sent from Slave program

Receive/unpack FN and 'who' from the 4 processors one by one

```
msgtype = 2
do 100 i=1,nproc
   call pvmrecv(-1,msgtype,info)
   call pvmfunpack(INTEGER4,who,1,1,info)
   call pvmfunpack REAL4,FN(who),1,1,info)
c continue
WRITE(10,*),'A ',A(1),A(2),A(3),A(4)
WRITE(10,*),'F ',FN(1),FN(2),FN(3),FN(4)
WRITE(10,*),' 'ERR = ABS(FN(2)-FN(3))
IF(ERR.LE.TOL) GOTO 999
C IF(FN(4) .GT. FN(3)) THEN
   B1 = A(3)
   B2 = A(2)
   A(1) = B1
   A(2) = B2
   GOTO 10
ELSE IF(FN(4) .LT. FN(3)) THEN
   B1 = A(1)
   B2 = A(4)
   A(1) = B1
   A(2) = B2
   GOTO 10
ENDIF
C Program finished leave PVM before exiting
999 continue
call pvmfexit(info)
STOP
END
```
PVM Slave Guideline

The Slave program is basically the function evaluation program. In order to do the function evaluation, it needs information from Master. For example, it needs the identity numbers \((tids(1), \ldots, tids(4))\) that PVM assigns, and the values of \(a_1, \ldots, a_4\).

The following steps are general guidelines to writing a Slave program. Please study the steps, and compare them with the program on the opposite page. Also try to find the connection with the Master Program. You may find Figure 1 helpful.

1. **Include fpvm3.h**
   Include this file in your program, you are able to use the PVM preset variable names; such as `PVMDEFAULT`, `REAL4`, and more, mentioned in all tables in Chapter 4 and the Appendix.

2. **Enroll Slave with PVM**
   Use `pvmfmytid(mytid)` to enroll.

3. **Identify the parent of this process**
   Use the following call to obtain the task identifiyer \((mtid)\) of parent process. This is useful for returning solutions to the Master. `pvmfparent(mtid)`

4. **Receive and Unpack data**
   Make sure the value of `msgtype` matches the one from Master. The sequence of unpacking is the same as the packing in Master.

5. **Perform function evaluation**

6. **Initialize buffer and pack data**
   Use `pvmfinitsend` to clear buffer.
   Use the following call to pack a real array \(F\) of dimension \(n\).
   `pvmfpack(REAL4,F,n,1,info)`

7. **Send data**
   Use the following call to send the packed message to Master:
   `pvmfsend(mtid,msgtag,info)`

8. **Exit PVM**
   Use `pvmfexit(info)` to exit PVM.
Slave Program

```
program function
include '../include/fpvm3.h'
integer tids(0:32), who
real a(32)
tor = 1.e-3

Enroll this program in PVM
call pvmfmytid(mytid)
Get the parent's task id
call pvmfparent(mtid)
continue
Receive data from host
msgtype = 1
call pvmfrecv(mtid, msgtype, info)
call pvmfunpack(INTEGER4, nproc, 1, l, info)
call pvmfunpack(INTEGER4, tids, nproc, 1, info)
call pvmfunpack(REAL4, A, 4, l, info)
call pvmfunpack(REAL4, ERR, 1, l, info)
if (err.le.tor) go to 99
Determine which processor I am
do 5 i=0,nproc-1
   if (tids(i).eq.mytid) me = i
   continue
who = me + 1
Calculate the function
X = A(who)
f = TANH(X)/(1.+X*X)
Send the result to Master
call pvmfinitsend(PVMDEFAULT, info)
call pvmfpack(INTEGER4, who, l, info)
call pvmfpack(REAL4, f, 1, l, info)
msgtype = 2
call pvmfsend(mtid, msgtype, info)
go to 3
Program finished. Leave PVM before exiting
continue
stop
end
```
Compilation and Running

After you finish your program, it is time to compile and run. Follow the steps below to compile your programs.

1. Make sure you have the correct directory setup
   Follow the advice from Directory Setup in Chapter 1.

2. Compile the program
   Use the sample Makefile on the opposite page to compile your programs.

   Note: The Makefile links the PVM library, `libfpvm3.a`.

3. Copy executables to all the hosts
   Follow the advice from Directory Setup in Chapter 1, and distribute the executables to `$HOME/pvm3/bin/ARCH`.

4. Activate PVM
   Activate PVM by entering `pvm` at UNIX prompt.

5. Decide the configuration of the virtual machine
   Add or delete hosts to the virtual machine. (Chapter 3)

6. Quit PVM console
   Leave PVM console (don’t halt daemon) by entering `quit` at the `pvm` prompt.
Makefile

# Custom section
# Set PVM_ARCH to your architecture type (SUN4, HP9K, RS6K, SGI, etc.)
# if PVM_ARCH = BSD386 then set ARCHLIB = -lrpc
# if PVM_ARCH = SGI then set ARCHLIB = -lsun
# if PVM_ARCH = I860 then set ARCHLIB = -lrpc -lssocket
# if PVM_ARCH = IPSC2 then set ARCHLIB = -lrpc -lssocket
# otherwise leave ARCHLIB blank
#
# PVM_ARCH and ARCHLIB are set for you if you use 'aimk'.
#
PVM_ARCH = SGI
ARCHLIB = -lsun
#
# END of custom section - leave this line here
#
PVMDIR = /amd/fs02/pub/iris4d_irix4/nas/pkg/pvm3.2
PVMLIB = $(PVMDIR)/lib/$(PVM_ARCH)/libpvm3.a
SDFR = 
BDIR = /u/wk/cheung/pvm3/bin
XDIR = $(BDIR)/$(PVM_ARCH)
CFLAGS = -g -I./include
LIBS = $(PVMLIB) $(ARCHLIB)
F77 = f77
FFLAGS = -g
FLIBS = $(PVMDIR)/lib/$(PVM_ARCH)/libfpvm3.a $(LIBS)

default: master function

$(XDIR):
    - mkdir $(BDIR)
    - mkdir $(XDIR)

clean:
    rm -f *.o bfgs quadfunct

master: $(SDFR)/master.f $(XDIR)
   $(F77) $(FFLAGS) -o master $(SDFR)/master.f $(FLIBS)
   mv master $(XDIR)

function: $(SDFR)/function.f $(XDIR)
   $(F77) $(FFLAGS) -o function $(SDFR)/function.f $(FLIBS)
   mv function $(XDIR)
Problems and Tips

PVM is a relatively new piece of software. It is not advanced enough to warn you ahead of time before problems come. Here are a couple of cases that you may encounter as a beginner.

Problems

Can’t activate PVM
- If the message you get, after entering `pvm` at UNIX prompt, is
  `libpvm [pid-1]: Console: Can’t start pvm`, it is possible that the last time you halted PVM daemon, the daemon created a residual file `/tmp/pvmd.xxxx`, where `xxxx` is an unique number for you. Delete this file, and start PVM again.
- If the daemon is running but the PVM console will not start, it is possible that you have too many processes running. You have to kill all the processes before you re-activate PVM console.

Note Use `ps -ef | username` at UNIX prompt to locate your running processes.

Can’t add hosts
It is possible that there are no links between your local computer and the other hosts. Check the following two things:
- Make sure each of your hosts has a `.rhosts` file in the `$HOME` directory, and this file points to your local computer.
- Make sure the `.rhosts` file is “read” and “write” protected from others users.
Problems and Tips

Host File

You can create the following file to build the virtual machine without activating the PVM console. The addresses must be recognizable by your system.

```
computer1.address
computer2.address
computer3.address
computer4.address
```

- host file

Note: The first machine listed must be the initiating host.

Note: If tasks are to be spawned on specific systems, the system name contained in `where` (routine `pvm_spawn`) must match the name in the host file exactly.

Note: If spawning tasks are on the initiating host, use the truncated host name. For example, if the full address is `win210.nas.nasa.gov`, use `win210` instead. This is a bug in PVM v3.2.

Having the host file ready, enter the following at UNIX prompt,

```
win210> pvmd3 host
```
Problems and Tips

Place to jot down problems.

Notes

If encounter problems, please contact:

Merritt Smith: mhsmith@nas.nasa.gov

or

Samson Cheung: cheung@nas.nasa.gov
### Appendix

#### TABLE 1. ARCH names used in PVM.

<table>
<thead>
<tr>
<th>ARCH</th>
<th>Machine</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFX8</td>
<td>Alliant FX 8</td>
<td>DEC OSF-1</td>
</tr>
<tr>
<td>ALPHA</td>
<td>DEC Alpha</td>
<td>DYNIX</td>
</tr>
<tr>
<td>BAL</td>
<td>Sequent Balance</td>
<td></td>
</tr>
<tr>
<td>BFLY</td>
<td>BBN Butterfly TC2000</td>
<td></td>
</tr>
<tr>
<td>BSD386</td>
<td>80386/486 Unix box</td>
<td>BSDI</td>
</tr>
<tr>
<td>CM2</td>
<td>Thanking Machines CM2</td>
<td>Sun front-end</td>
</tr>
<tr>
<td>CM5</td>
<td>Thanking Machines CM5</td>
<td></td>
</tr>
<tr>
<td>CNVX</td>
<td>Convex C-series</td>
<td></td>
</tr>
<tr>
<td>CNVXN</td>
<td>Convex C-series</td>
<td>native mode</td>
</tr>
<tr>
<td>CRAY</td>
<td>C-90, YMP, Cray-2</td>
<td>UNICOS</td>
</tr>
<tr>
<td>CRAYSMP</td>
<td>Cray S-MP</td>
<td></td>
</tr>
<tr>
<td>DGAV</td>
<td>Data General Aviion</td>
<td></td>
</tr>
<tr>
<td>HP300</td>
<td>HP-9000 model 300</td>
<td>HPUX</td>
</tr>
<tr>
<td>HPPA</td>
<td>HP-9000 PA-RISC</td>
<td></td>
</tr>
<tr>
<td>IB60</td>
<td>Intel IPSC/860</td>
<td>link-lprc</td>
</tr>
<tr>
<td>IPSC2</td>
<td>Intel iPSC/860 host</td>
<td>SysV</td>
</tr>
<tr>
<td>KSRI</td>
<td>Kendall Square KSR-1</td>
<td>OSF-1</td>
</tr>
<tr>
<td>NEXT</td>
<td>NeXT</td>
<td></td>
</tr>
<tr>
<td>PGON</td>
<td>Intel Paragon</td>
<td>link-lprc</td>
</tr>
<tr>
<td>PMAX</td>
<td>DECstation 3100,5100</td>
<td>Ultrix</td>
</tr>
<tr>
<td>RS6K</td>
<td>IBM/RS6000</td>
<td>AIX</td>
</tr>
<tr>
<td>RT</td>
<td>IBM RT</td>
<td></td>
</tr>
<tr>
<td>SGI</td>
<td>Silicon Graphics IRIS</td>
<td>link-lsrun</td>
</tr>
<tr>
<td>SUN3</td>
<td>Sun 3</td>
<td>SunOS</td>
</tr>
<tr>
<td>SUN4</td>
<td>Sun 4, SPARCstation</td>
<td></td>
</tr>
<tr>
<td>SYMM</td>
<td>Sequent Symmetry</td>
<td></td>
</tr>
<tr>
<td>TITN</td>
<td>Staedent Titan</td>
<td></td>
</tr>
<tr>
<td>UVAX</td>
<td>DEC Micro VAX</td>
<td></td>
</tr>
</tbody>
</table>
### TABLE 2. Error codes returned by PVM routines

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PvmOK (0)</td>
<td>All right</td>
</tr>
<tr>
<td>PvmBadParam (-2)</td>
<td>Bad parameter</td>
</tr>
<tr>
<td>PvmMismatch (-3)</td>
<td>Barrier count mismatch</td>
</tr>
<tr>
<td>PvmNoData (-5)</td>
<td>Read past end of buffer</td>
</tr>
<tr>
<td>PvmNoHost (-6)</td>
<td>No such host</td>
</tr>
<tr>
<td>PvmNoFile (-7)</td>
<td>No such executable</td>
</tr>
<tr>
<td>PvmNoMem (-10)</td>
<td>Can't get memory</td>
</tr>
<tr>
<td>PvmBadMsg (-12)</td>
<td>Can't decode received message</td>
</tr>
<tr>
<td>PvmSysErr (-14)</td>
<td>Pvmd not responding</td>
</tr>
<tr>
<td>PvmNoBuf (-15)</td>
<td>No current buffer</td>
</tr>
<tr>
<td>PvmNoSuchBuf (-16)</td>
<td>Bad message identifier</td>
</tr>
<tr>
<td>PvmNukkGroup (-17)</td>
<td>Null group name is illegal</td>
</tr>
<tr>
<td>PvmDupGroup (-18)</td>
<td>Already in group</td>
</tr>
<tr>
<td>PvmNoGroup (-19)</td>
<td>No group with that name</td>
</tr>
<tr>
<td>PvmNotInGroup (-20)</td>
<td>Not in group</td>
</tr>
<tr>
<td>PvmNoInst (-21)</td>
<td>No such instance in group</td>
</tr>
<tr>
<td>PvmHostFail (-22)</td>
<td>Host failed</td>
</tr>
<tr>
<td>PvmNoParent (-23)</td>
<td>No parent task</td>
</tr>
<tr>
<td>PvmNoImpl (-24)</td>
<td>Function not implemented</td>
</tr>
<tr>
<td>PvmDSysErr (-25)</td>
<td>Pvmd system error</td>
</tr>
<tr>
<td>PvmBadVersion (-26)</td>
<td>Pvmd-pvmid protocol mismatch</td>
</tr>
<tr>
<td>PvmOutOfRes (-27)</td>
<td>Out of resources</td>
</tr>
<tr>
<td>PvmDupHost (-28)</td>
<td>Host already configured</td>
</tr>
<tr>
<td>PvmCantStart (-29)</td>
<td>Fail to execute new slave pvmid</td>
</tr>
<tr>
<td>PvmAlready (-30)</td>
<td>Slave pvmid already running</td>
</tr>
<tr>
<td>PvmNoTask (-31)</td>
<td>Task does not exist</td>
</tr>
<tr>
<td>PvmNoEntry (-32)</td>
<td>No such (group,instance)</td>
</tr>
<tr>
<td>PvmDupEntry (-33)</td>
<td>(Group,instance) already exists</td>
</tr>
</tbody>
</table>