Colorado State University
Ft. Collins, CO 80523

and

Institute for Computational Studies
P.O. Box 1852
Ft. Collins, CO 80522

(NASA-CR-177020) REMOTE ACCESS FOR NAS: SUPERCOMPUTING IN A UNIVERSITY ENVIRONMENT Final Report, 1 Apr. 1985 - 31 Mar. 1986 (Colorado State Univ.) 60 p CSCL 09B
N86-30370
Unclas
G3/61 43229

Remote Access for NAS: Supercomputing in a University Environment

Final Report, covering the period April 1, 1985 through March 31, 1986

Cooperative Agreement NCC2-344

Gary Johnson - Principal Investigator
Becky Olson
Julie Swisshelm
Dan Pryor
John Ziebarth

The experiment was designed to assist the NAS (Numerical Aerodynamic Simulation) Project Office in the testing and evaluation of long haul communications for remote users. The objectives of this work were to:

(1) Use foreign workstations to remotely access the NAS system. In this way problems associated with interfacing equipment commonly used by university and industrial investigators to the NAS facility can be identified and solutions to these problems found.

(2) Provide NAS with a link to a large university-based computing facility which can serve as a model for a regional node of the Long-Haul Communications Subsystem (LHCS).

(3) Provide a tail circuit to the University of Colorado at Boulder thereby simulating the complete communications path from NAS through a regional node to an end-user.

To meet these objectives the Institute for Computational Studies (ICS) took delivery of a Sun-2 160 color workstation in the first part of June. Another Sun-2 160 color workstation arrived in early July.

The first problem encountered by NAS/ICS was the inability of the Sun workstation to run a graphics software package (PLOT3D) available on NASA's Silicon Graphics IRIS workstation. Becky Olson spent a week at NASA/Ames in May learning about the graphics package and the IRIS. She spent the next month converting it to the SUN workstation so the communication experiment could take place. The modified source code for the Sun version of PLOT3D is listed in Appendix I.

The 56 kilobaud line and the data service unit were installed during the week of July 29. By August 1, NASA/Ames and ICS were communicating through the 56 kilobaud line, providing NAS with a link to a university.

The next phase of the experiment was to determine if the 56 kilobaud line was adequate for actual development work and data and graphic file transfers. Julie Swisshelm spent the week of August 19 through 23 conducting Computational Fluid Dynamics (CFD) runs at NASA/Ames to have a base in determining the time it takes to run a CFD problem and transfer files locally at NASA Ames.

The next week (August 26 through August 30) the CFD runs were done at ICS through the 56 kilobaud line with statistical data accumulated to see if the line speed was acceptable. This data is being processed by NASA/Ames to evaluate the difference between on-site users and remote users. In September, ICS took delivery of a Silicon Graphics IRIS workstation. This was added to our local area network and used also to access the Cray X-MP/12 and Cray-2 at NASA/Ames.

By October most of the problems associated with the long haul communications experiment, such as the instability of Vitalink and the response time of Amelia, had been fixed and a test production mode was begun.

During the next few months, an efficient algorithm (CNS3D) to be used for Navier-Stokes simulations of complex aircraft and turbomachinery geometries, was developed. The paper describing the results of this algorithm development has been accepted for presentation in the Tenth International Conference on Numerical Methods in Fluid Dynamics to be held in Beijing, China on June 23-27,1986. Included as Appendix II to this report is the extended abstract for this conference. A code to do the embedded multigridding for use in the algorithm was developed on the Sun workstations/CSU Cyber 205 for a rectilinear cascade of finite-span, swept blades mounted between endwalls. This code was made operational on the NAS system. This allowed the code to use the large memory capability so that realistically fine meshes for this current turbomachinery test geometry can be generated and will allow extensions of the code to include complete aircraft and turbomachinery geometries. Other code was developed and old code converted to run on the Cray-2. So far we have converted a library of i/o routines that we have found helpful in debugging code on both the Cray XMP and the CYBER 205. Unfortunately, all of them use features not currently supported by the Cray-2 FORTRAN compiler. In addition to that library, a set of routines that interpolate efficiently between grid levels in multiple grid problems has been developed, with the Cray-2 as the target machine. That is, we have been careful to build into the code several parameters that can be adjusted if the long memory access time becomes a problem. A code to do three-dimensional incompressible viscous calculations has also been implemented on the Cray-2.

The first two objectives of the experiment have been met. We discovered that the researchers preferred to do most of the editing on the local workstations and ship the code over as opposed to doing it at the remote site. This was due to the familiarity of the editor on the local workstation and the response time and convenience of a full screen editor. Utilities were created that allowed the researchers to bypass any interaction with Amelia and move files directly from the local workstation to the Cray-2, and from the Cray-2 to our local workstations. These utilities are available from ICS, on request. The researchers preferred to use the local workstations and just submit files to the Cray-2. However, this preference could switch if a user friendly full screen editor were available on the Cray-2. It was discovered that 56 kilobaud was an adequate line speed for transferring source codes and small to medium size data sets. For very large data sets 56 kilobaud can create very long transfer times. Our timings and calculations demonstrate a 56 kilobaud connection to Amelia takes about 8 minutes to transfer a 3,055,850 byte file from our SUN to Amelia using the command 'rcp'. The command 'ftp' is much slower, taking 11 minutes to transfer the same size file. Using rcp and the 'to2' command to transfer that same file from our SUN to the Cray-2 takes 11 minutes. Finally, the researchers preferred to generate the graphics files on the workstations as opposed to generating them on the remote machine. This was due in part to the size of a meta file that would have to be shipped back and also the interactive capability of changing the characteristics of the graphics file at the local workstation.

The third objective, that of providing a tail circuit to Boulder, has not yet been accomplished. Completion of this circuit has been delayed by slippage in the schedules for LAN interconnections at both CU Boulder and CSU and by slippage in the schedule for the reconfiguration of the Colorado Advanced Technology Institute (CATI) funded 56Kb link between the CU Boulder and CSU computing centers. To date, we have no indication that these delays are anything more than normal schedule slippage. Work is continuing, beyond the period of performance of the present grant, and should be complete by mid-summer.

Recommendations

A couple of items would enhance the convenience of the current configuration. First, it would be helpful at times to be able to submit batch jobs to the Cray-2 from the local workstations, and have the job output automatically returned to the local workstation or LAN, without the user having to log into any other machine. This would be particularly helpful when doing production runs, where there is no real need to be interactively logged into the Cray-2. We developed this capability for the NAS Project Cray X-MP 12, and found it to be a very valuable tool. Incidentally, such a situation would also relieve the problem of line contention on the Cray-2 which, though not critical now, may become so in time. On the workstation side, a job queuing mechanism would be nice, so that jobs could be submitted to the Cray-2, and eventually run, even when one or more of the links in the path to the machine are temporarily down.

Second, we realize that the software on the Cray-2 is as yet unfinished, but we feel that in time the machine environment ought to be made entirely compatible with that of other Cray models. The need for total portability of FORTRAN code and a Cray compatible version of Update are noteworthy. There are products on the market, such as Historian from Opcode Inc., that emulate and extend the features of Update. Perhaps the NAS project could develop a version of Update, or at least an Update subset that would handle Update source files from other Cray environments.

Appendix I

```
/*
  disspla lookalike library
*/
#include <usercore.h>
#include <sun/fbio.h>
#include "disspla.h"

/* ----------------------------------------*/
sun2_()
/*
  initialize sun core and screen
*/

{
    set_up_core2();
/*
 set up random generator for segment numbers
*/
    srand(1);
/*
 set up color table
*/
    define_color_indices(&vsurf, 0, MAPSIZE, red, green, blue);
}
/* ----------------------------------------*/
sun3_()
/*
  initialize sun core and screen
*/

{
    int i;
    extern TWODIM;
    set_up_core3();
    TWODIM = FALSE;
/*
 set up random generator for segment numbers
*/
    srand(1);
/*
 set up color table
*/
    define_color_indices(&vsurf, 0, MAPSIZE, red, green, blue);
}
/* ----------------------------------------*/
set_up_core2()
{
  extern struct vwsurf vsurf;

    get_view_surface(&vsurf,NULL);
    vsurf.cmapsize = MAPSIZE;
    vsurf.cmapname[0] = '\0';
/*
  pump to full capability
*/
    if (initialize_core(DYNAMICC, SYNCHRONOUS, TWOD)) exit(0);
```

```
    if (initialize_view_surface(&vsurf, FALSE)) exit(0);
/*
  set ndc to default
*/
    set_ndc_space_2(1.0,0.75);
/*
  initialize devices
*/
    initdev_();
/*
  set clipping capability
*/
    set_window_clipping(TRUE);
    set_output_clipping(TRUE);
    select_view_surface(&vsurf);
/*
  set up transformation types
*/
    set_image_transformation_type(XFORM2);
}
/* ----------------------------------------*/
set_up_core3()
{
  extern struct vwsurf vsurf;

    get_view_surface(&vsurf,NULL);
    vsurf.cmapsize = MAPSIZE;
    vsurf.cmapname[0] = '\0';
/*
  pump to full capability
*/
    if (initialize_core(DYNAMICC, SYNCHRONOUS, THREED)) exit(0);
    if (initialize_view_surface(&vsurf, FALSE)) exit(0);
/*
  initialize devices
*/
    initdev_();
/*
  set clipping capability
*/
    set_window_clipping(TRUE);
    set_output_clipping(TRUE);
    select_view_surface(&vsurf);
/*
  set transformation capability
*/
    set_image_transformation_type(XFORM3);
}
/* ----------------------------------------*/
initdev_()
{

/*
  initialize input devices
*/
    initialize_device( BUTTON, 1);            /* initialize input devices */
```

```
    initialize_device( BUTTON, 2);              /* initialize input devices */
    initialize_device( BUTTON, 3);              /* initialize input devices */
    initialize_device( LOCATOR, 1);
    initialize_device( PICK, 1);
    set_echo_position( LOCATOR, 1, 0., 0.);
    set_echo_surface( LOCATOR, 1,&vsurf);
    set_echo_surface( PICK, 1, &vsurf);
    set_pick(1,0.001);
}
/* ----------------------------------------*/
graf3d_(x3min,x3stp,x3max,y3min,y3stp,y3max,z3min,z3stp,z3max)
/*
   3d window set, no axis drawn on 3d
*/
float *x3min,*x3stp,*x3max;
float *y3min,*y3stp,*y3max;
float *z3min,*z3stp,*z3max;

{
  float xndcchar1, xndcchar2, yndcchar1, yndcchar2;
  float xwldchar1, xwldchar2, ywldchar1, ywldchar2;
  float zndcchar1, zndcchar2;
  float zwldchar1, zwldchar2;
  float xndc, yndc, xct, yct;

      xmin = *x3min;
      xstp = *x3stp;
      xmax = *x3max;
      ymin = *y3min;
      ystp = *y3stp;
      ymax = *y3max;
      zmin = *z3min;
      zstp = *z3stp;
      zmax = *z3max;
      zndcchar1 = 0.;
      zndcchar2 = 0.;
      set_viewport_3(.0,1.,.0,.75,.0,.75);
      xct = (xmax - xmin) / 10.;
      yct = (ymax - ymin) / 10.;
      set_window(xmin-xct,xmax+xct,ymin,ymax+yct);
      xndcchar1 = .2;
      yndcchar1 = .2;
      xndcchar2 = .27;
      yndcchar2 = .27;
      map_ndc_to_world_3(xndcchar1,yndcchar1,zndcchar1,
                  &xwldchar1,&ywldchar1,&zwldchar1);
      map_ndc_to_world_3(xndcchar2,yndcchar2,zndcchar2,
                  &xwldchar2,&ywldchar2,&zwldchar2);
      charwiddef = xwldchar2 - xwldchar1;
      charheidef  = ywldchar2 - ywldchar1;
      if (charwiddef < 0)
         charwiddef = -1 * charwiddef;
      if (charheidef < 0)
         charheidef = -1 * charheidef;
      charheinew = charheidef;
      charwidnew = charwiddef;
```

```
/*
  set type of character ie font and set height and width for
  labels
*/
      set_charprecision(CHARACTER);
      set_font(STICK);
      set_charsize(charwiddef,charheidef);
}
/* ----------------------------------------*/
ftoa_(r,whole)
/*
   change from a real number to character
*/

float r;
char whole[20];

{
   float rem, fnum;
   int num, neg, cnt;
   int i;
   char c;
   char str[20];

    if (r < 0.) {
        neg = TRUE;
        r *= -1.;
    }
    else
        neg = FALSE;
    cnt = 0;
    num = i = r;
    fnum = i;
    rem = r - fnum;
    if (rem > 0.)
        str[cnt++] = '.';
    while ( rem > 0.) {
        rem *= 10.;
        i = rem;
        itoa_(i,&c);
        str[cnt++] = c;
        fnum = i;
        rem -= fnum;
    }
    str[cnt] = '\0';
    if (neg) {
        whole[0] = '-';
        itoa_(num,&whole[1]);
    }
    else {
        whole[0] = ' ';
        itoa_(num,&whole[1]);
    }
    strcat(whole,str);
}
/* ----------------------------------------*/
```

```
itoa_(n,sti)
/*
  change integer to a string
*/

char sti[];
int n;

{
   int i, sign;

   if ((sign = n) < 0 )
        n = n;
   i = 0;
   do {
        sti[i++]  = n % 10 + '0';
   } while ((n /= 10) > 0);
   if (sign < 0)
       sti[i++] = '-';
   sti[i] = '\0';
   reverse(sti);
}
/* ----------------------------------------*/
reverse(srev)
/*
  reverse the order of a string
*/

char srev[];

{
   int c, i, j;

   for (i = 0, j = strlen(srev) -1; i < j; i++, j--) {
        c = srev[i];
        srev[i] = srev[j];
        srev[j] = c;
   }
}
/* ----------------------------------------*/
messag_(string,xpos,ypos)
/*
  put a text string on screen starting at xpos, ypos
*/
char string[];
float *xpos, *ypos;

{
   float x, y;

   x = *xpos;
   y = *ypos;

   set_charsize(charwiddef,charheidef);
   move_abs_2(x,y);
   text(string);
```

```
}
/* ----------------------------------------*/
headin_(title,hei,num)
/*
  write a title or label, letting the mouse
  prompt for the position
*/

char title[];
int *num;
float *hei;

{
   float xpos, ypos;
   float xndc, yndc;
   float xs,xe;
   float heig;
   int butnum;
   heig = *hei;
/*
  set character type to character do can be manipulated ie
  rotated etc
*/
   set_charprecision(CHARACTER);
   set_font(STICK);
   charwidnew = charwiddef * heig;
   charheinew = charheidef * heig;
   set_charsize(charwidnew,charheinew);
/**/
   butnum = 0;
/*
  wait for mouse click to determine location
*/
   set_echo(LOCATOR,1,1);
   printf(" click a mouse button at the position the title should begin\n");
   do {
       await_any_button_get_locator_2(10000000,1,&butnum,&xndc,&yndc);
   }  while (butnum < 1 || butnum > 3);
/*
  write text
*/
   map_ndc_to_world_2(xndc,yndc,&xpos,&ypos);
   move_abs_2(xpos,ypos);
   text(title);
/*
  reset character size
*/
   set_charsize(charwiddef,charheidef);
}
/* ----------------------------------------*/
x3name_(title,len)
/*
  print x axis label
*/

char title[];
```

```
int *len;

{
   float xpos, ypos;
   float xndc, yndc;
   float xs,xe;
   int titlen;
   int butnum;


   titlen = *len;
/*
  set charsize
*/
   set_charsize(charwidnew,charheinew);
   charwidnew = charwiddef;
   charheinew = charheidef;
/*
   create segment for x axis label
*/
   segnum = rand();
   create_retained_segment(segnum);
   butnum = 0;
/*
  wait for mouse button for location
*/
   set_echo(LOCATOR,1,1);
   printf(" click a mouse button at the position the x label should begin\n");
   do {
       await_any_button_get_locator_2(10000000,1,&butnum,&xndc,&yndc);
   }  while (butnum < 1 || butnum > 3);
/*
  write out text
*/
   map_ndc_to_world_2(xndc,yndc,&xpos,&ypos);
   move_abs_2(xpos,ypos);
   text(title);
   close_retained_segment();
   set_charsize(charwiddef,charheidef);
}
/* ----------------------------------------*/
y3name_(title,len)
/*
  print y axis label
*/

char title[];
int *len;

{
   float xpos, ypos;
   float xndc, yndc;
   float xs,xe;
   int titlen;
   int butnum;
```

```
   titlen = *len;
   set_charsize(charheinew,charwidnew);
   charwidnew = charwiddef;
   charheinew = charheidef;
/*
  create segment for y axis
*/
   segnum = rand();
   create_retained_segment(segnum);
/*
  make sure it is going up y axis
*/
   set_charup_2(-1.,0.);
   set_charpath_2(0.,1.);
   butnum = 0;
/*
  wait for mouse button for location
*/
   set_echo(LOCATOR,1,1);
   printf(" click a mouse button at the position the y label should begin\n");
   do {
       await_any_button_get_locator_2(10000000,1,&butnum,&xndc,&yndc);
   }  while (butnum < 1 || butnum > 3);
/*
  write out text
*/
   map_ndc_to_world_2(xndc,yndc,&xpos,&ypos);
   move_abs_2(xpos,ypos);
   text(title);
/*
  revert character direction back to normal
*/
   set_charup_2(0.,1.);
   set_charpath_2(1.,0.);
   close_retained_segment();
   set_charsize(charwiddef,charheidef);
}
/* ----------------------------------------*/
z3name_(title,len)
/*
  print y axis label
*/

char title[];
int *len;

{
   float xpos, ypos;
   float xndc, yndc;
   float xs,xe;
   int titlen;
   int butnum;


   titlen = *len;
```

```
   set_charsize(charheinew,charwidnew);
   charwidnew = charwiddef;
   charheinew = charheidef;
/*
  create segment for y axis
*/
   segnum = rand();
   create_retained_segment(segnum);
/*
  make sure it is going up y axis
*/
   set_charup_2(-1.,0.);
   set_charpath_2(0.,1.);
   butnum = 0;
/*
  wait for mouse button for location
*/
   set_echo(LOCATOR,1,1);
   printf(" click a mouse button at the position the y label should begin\n");
   do {
       await_any_button_get_locator_2(10000000,1,&butnum,&xndc,&yndc);
   }  while (butnum < 1 || butnum > 3);
/*
  write out text
*/
   map_ndc_to_world_2(xndc,yndc,&xpos,&ypos);
   move_abs_2(xpos,ypos);
   text(title);
/*
  revert character direction back to normal
*/
   set_charup_2(0.,1.);
   set_charpath_2(1.,0.);
   close_retained_segment();
   set_charsize(charwiddef,charheidef);
}
/* ------------------------------------------*/
curve_(xaray, yaray, npnts, imark)
/*
 actually plots the data, ie xaray by yaray
*/
float *xaray, *yaray;
int *imark, *npnts;

{

   float xval[NUMELEMENTS], yval[NUMELEMENTS];
   int num, mark, posmark;
   int sym;
   int i, j;

   num = *npnts;
   mark = *imark;

   set_window_clipping(TRUE);
   set_output_clipping(TRUE);
```

```
    posmark = mark;
    if (mark < 0 )
        posmark *= -1;
    for (i = 0; i < num+1; i++) {
         xval[i] = *xaray++;
         yval[i] = *yaray++;
    }

    segnum = rand();
    create_retained_segment(segnum);
    if (mark != 0) {
        if (!SETSYMBOL) {
            inquire_marker_symbol(sym);
            sym += 1;
            if (sym > 18)
               sym = 1;
            j = marksym[sym-1];
            set_marker_symbol(j);
        }
        for (i = 0; i < num+1; i += posmark)
              marker_abs_2(xval[i],yval[i]);
        SETSYMBOL = FALSE;
    }
    move_abs_2(xval[0],yval[0]);
    if (mark >= 0)
        polyline_abs_2(xval,yval,num);
    close_retained_segment();
}
/* ----------------------------------------*/
line3_(npnts, xaray, yaray, zaray)
/*
 actually plots the data, ie xaray by yaray
*/
float *xaray, *yaray, *zaray;
int *npnts;

{

    float xval[NUMELEMENTS], yval[NUMELEMENTS], zval[NUMELEMENTS];
    int num;
    int sym;
    int i, j;

    num = *npnts;

    for (i = 0; i < num+1; i++) {
         xval[i] = *xaray++;
         yval[i] = *yaray++;
         zval[i] = *zaray++;
    }

    move_abs_3(xval[0],yval[0],zval[0]);
    polyline_abs_3(xval,yval,zval,num);
}

/* ----------------------------------------*/
```

```
line2_(npnts, xaray, yaray)
/*
 actually plots the data, ie xaray by yaray
*/
float *xaray, *yaray;
int *npnts;

{

   float xval[NUMELEMENTS], yval[NUMELEMENTS];
   int num;
   int sym;
   int i, j;

   num = *npnts;

   for (i = 0; i < num+1; i++) {
        xval[i] = *xaray++;
        yval[i] = *yaray++;
   }

   move_abs_2(xval[0],yval[0]);
   polyline_abs_2(xval,yval,num);
}
/* ----------------------------------------*/
rlvec_(xpnt1, ypnt1, xpnt2, ypnt2, ivec)
/*
 actually plots a line
*/
float *xpnt1, *ypnt1;
float *xpnt2, *ypnt2;
int *ivec;

{

   float xfrom, yfrom, xto, yto;
   int num, mark, posmark;
   int isym;


   xfrom = *xpnt1;
   yfrom = *ypnt1;
   xto = *xpnt2;
   yto = *ypnt2;

   segnum = rand();
   create_retained_segment(segnum);
   move_abs_2(xfrom,yfrom);
   line_abs_2(xto,yto);
   close_retained_segment();
}
/* ----------------------------------------*/
vecto3_(xpnt1, ypnt1, zpnt1, xpnt2, ypnt2, zpnt2)
/*
 actually plots a 3-d line
*/
```

```
float *xpnt1, *ypnt1, *zpnt1;
float *xpnt2, *ypnt2, *zpnt2;

{

   float xfrom, yfrom, zfrom, xto, yto, zto;
   int arrsym;


   xfrom = *xpnt1;
   yfrom = *ypnt1;
   zfrom = *zpnt1;
   xto = *xpnt2;
   yto = *ypnt2;
   zto = *zpnt2;

   arrsym = 62;
   move_abs_3(xfrom,yfrom,zfrom);
   line_abs_3(xto,yto,zto);
}
/* ------------------------------------------*/
object_(array)
/*
  open a segment
*/
char *array;

{

   segnum = rand();
   create_retained_segment(segnum);

}
/* ------------------------------------------*/
endobj_()
/*
  close a segment
*/

{

  popatt_();
  close_retained_segment();

}
/* ------------------------------------------*/
strtpt_(xpnt1, ypnt1)
/*
 plots a point
*/
float *xpnt1, *ypnt1;

{
   float xfrom, yfrom;
   xfrom = *xpnt1;
   yfrom = *ypnt1;
```

```
   move_abs_2(xfrom,yfrom);
}
/* ----------------------------------------*/
strtpt3_(xpnt1, ypnt1, zpnt1)
/*
 plots a point
*/
float *xpnt1, *ypnt1, *zpnt1;

{
   float xfrom, yfrom, zfrom;
   xfrom = *xpnt1;
   yfrom = *ypnt1;
   zfrom = *zpnt1;

   move_abs_3(xfrom,yfrom,zfrom);
}
/* ----------------------------------------*/
connpt_(xpnt1, ypnt1)
/*
 actually plots a line
*/
float *xpnt1, *ypnt1;

{
   float xfrom, yfrom;
   xfrom = *xpnt1;
   yfrom = *ypnt1;

   line_abs_2(xfrom,yfrom);
}
/* ----------------------------------------*/
connpt3_(xpnt1, ypnt1, zpnt1)
/*
 actually plots a line
*/
float *xpnt1, *ypnt1, *zpnt1;

{
   float xfrom, yfrom, zfrom;
   xfrom = *xpnt1;
   yfrom = *ypnt1;
   zfrom = *zpnt1;

   line_abs_3(xfrom,yfrom,zfrom);
}
/* ----------------------------------------*/
marker_(isym)
/*
  define a marker symbol
*/

int *isym;

{
```

```
  int sym, j;

  sym = *isym;

  SETSYMBOL = TRUE;
  j = sym - 1;
  set_marker_symbol(marksym[j]);
}
/* ----------------------------------------*/
endpl_(iplot)
/*
  end a plot nothing to do
*/

int *iplot;
{
/*   color_(1);
   clear_(); */
}
/* ----------------------------------------*/
donplt_()
/*
  totally finished plotting
*/
{
  extern struct vwsurf vsurf;

  deselect_view_surface(&vsurf);
  terminate_core();
}
/* ----------------------------------------*/
gethei_()
/*
  sets character size of text, default set in setup at .014
  of a inch
*/

{
  float hite2;

   hite2 = charheidef;

   return *((int*)&hite2);
}
/* ----------------------------------------*/
savescreen_(filenam,len)
/*
  save a screen in a file
*/

char filenam[];
int *len;

{
   struct  {
       int width, height, depth;
```

```
        short *bits;
    }  raster;

    struct  {
        int type;
        int nbytes;
        char *data;
    }  map;

    char filetit[20];
    int rasfid;
    int replicate;
    int strlen;
    int i;
    float wx, wy;

/*
  save color map
*/
/*   colmap[0] = &red[0];
    colmap[1] = &green[0];
    colmap[2] = &blue[0];
*/
    strlen = *len;
    for (i = 0; i <= strlen; i++)
        filetit[i] = filenam[i];
    filetit[i] = '\0';
    replicate = 2;
/*
  get starting location for save
*/
    map_ndc_to_world_2( .0, .0, &wx, &wy);
    move_abs_2(wx, wy);
/*
  make sure memory is allocated
*/
    size_raster(&vsurf,.0,1.,.0,.75,&raster);
    allocate_raster(&raster);
    if (raster.bits == NULL) {
       printf ("failed to allocate raster\n");
    }
    else {

       get_raster(&vsurf,.0,1.,.0,.75,0,0,&raster);
/*
  save it
*/
       if( (rasfid = open( filetit, 1)) == -1) {   /* open the disk file */
       }
       if (rasfid != -1) {
            map.type = 1;  map.nbytes = 0;
            raster_to_file( &raster, &map, rasfid, replicate);
            close( rasfid);
       }
       free_raster(&raster);
     }
```

```
}
/* ----------------------------------------*/
color_(index)
/*
  sets the color and fill and text
*/
int *index;

{
    int ind;

    ind = *index;
    set_line_index(ind);
    set_fill_index(ind);
    set_text_index(ind);
}
/* ----------------------------------------*/
settex_(index)
/*
  sets the color of the text
*/
int *index;
{
    int ind;

    ind = *index;
    set_text_index(ind);
}
/* ----------------------------------------*/
polfil_(index)
/*
  sets the color of the fill area
*/
int *index;
{
    int ind;

    ind = *index;
    set_fill_index(ind);
}
/* ----------------------------------------*/
clear_()
/*
  clear screen
*/
{
      delete_all_retained_segments();
      new_frame();
}
/* ----------------------------------------*/
linsty_(style)
/*
   set linestyle for the line ie
   style (dot dash etc )
*/
```

```
int *style;

{
   int linesty;

   linesty = *style;
   linesty = linesty - 1;

   if (linesty > 5)
       linesty = 0.;

   set_linestyle(linesty);
   linestyle = linesty;
}
/* ----------------------------------------*/
linwid_(thick)
/*
   set width for the line ie
*/

float *thick;

{
   float thickness;
   int width;

   thickness = *thick;
   if (thickness <= 1.)
       width = 0;
   else
       width = 1;

   set_linewidth(width);
}
/* ----------------------------------------*/
lineatt_(style,thick)
/*
   set attributes for the line ie
   style (dot dash etc and the thickness)
*/

float *style;
int   *thick;

{
   int thickness;
   float linestyle;

   linestyle = *style;
   thickness = *thick;

   if (linestyle > 3)
       linestyle = 0.;

   set_linestyle(linestyle);
```

```
   set_linewidth(thickness);
}
/* ----------------------------------------*/
setcol_(redval,grnval,bluval,loc)
/*
  sets color in rgb mode
*/
float *redval, *grnval, *bluval;
int *loc;

{
      float redcol, grncol, blucol;
      int ict;

      redcol = *redval;
      grncol = *grnval;
      blucol = *bluval;

      ict = *loc;

      red[ict] = redcol;
      green[ict] = grncol;
      blue[ict] = blucol;
}
/* ----------------------------------------*/
rgbcol_(r,g,b)
/*
  set a color and then change to it
*/
float *r, *g, *b;

{

  red[255] = *r;
  green[255] = *g;
  blue[255] = *b;

    define_color_indices(&vsurf, 0, MAPSIZE , red, green, blue);
}
/* ----------------------------------------*/
getrgb_(icol,rval,grnval,bluval)
/*
  get rgb values for index in color map
*/
float rval, grnval, bluval;
int *icol;

{
    int index;

    index = *icol;

    rval = red[index];
    grnval = green[index];
    bluval = blue[index];
}
```

```
/* ----------------------------------------*/
polsur_(proper,iproper)
/*
  set shading properties
*/
  float *proper[];
  int *iproper[];

{
   float ambient, diffuse, secular, flood, bump;
   int hue, style;

   ambient = *proper[0];
   diffuse = *proper[1];
   secular = *proper[2];
   flood   = *proper[3];
   bump    = *proper[4];
   hue     = 1;
   style   = *iproper[1];

}
/* ----------------------------------------*/
poly2_(num,xarr,yarr)
/*
  draw a polygon
*/
  float *xarr[], *yarr[];
  int *num;
{
  float x[], y[];
  int numb;

  x[0] = *xarr[0];
  y[0] = *yarr[0];

  numb = *num;

  polygon_abs_2(x,y,numb);
}
/* ----------------------------------------*/
polf_(num,xarr,yarr,zarr)
/*
  draw a polygon
*/
  float *xarr[], *yarr[], *zarr[];
  int *num;
{
  float x[], y[], z[];
  int numb;

  x[0] = *xarr[0];
  y[0] = *yarr[0];
  z[0] = *zarr[0];

  numb = *num;
```

```
  polygon_abs_3(x,y,z,numb);
}
/* ------------------------------- */
mapcol_(index,rdval,grnval,bluval)
/*
  create color map
*/
  int *index;
  float *rdval,*grnval,*bluval;
{
  float redrgb, grnrgb, blurgb;
  int count;

  count = *index;

  red[count]= *rdval;
  blue[count] = *bluval;
  green[count] = *grnval;
}
/* ------------------------------------- */
vecto2_(xstart,ystart,xend,yend)
/*
  draw a 2 d vector
*/

float *xstart, *ystart, *xend, *yend;

{
  float xs, ys, xe, ye;

  xs = *xstart;
  ys = *ystart;
  xe = *xend;
  ye = *yend;

  move_abs_2(xs,ys);
  line_abs_2(xe,ye);
}
/* ------------------------------------------ */
point3_(x,y,z)
/*
  make a point at the given coordinates
*/
float *x, *y, *z;
{
  float xpos, ypos, zpos;
  int dot;

  dot = 46;
  xpos = *x;
  ypos = *y;
  zpos = *z;

  set_marker_symbol(dot);
  marker_abs_3(xpos,ypos,zpos);
}
```

```
/* ------------------------------------------ */
point2_(x,y)
/*
  make a point at the given coordinates
*/
float *x, *y;
{
  float xpos, ypos;
  int dot;

  dot = 46;
  xpos = *x;
  ypos = *y;

  set_marker_symbol(dot);
  marker_abs_2(xpos,ypos);
}
pushma_()
{
}
popatt_()
{
}
tmpfile()
{
}
popmat_()
{
}
/* ------------------------------------------------ */
height_(num)
/*
  change height of numbers
*/
float *num;
{
  float multiple;

  multiple = *num;
  charheidef = charheidef * multiple;
  charwiddef = charwiddef * multiple;
}
```

Appendix II

ABSTRACT

MULTITASKED EMBEDDED MULTIGRID FOR THREE-DIMENSIONAL FLOW SIMULATION

by

Gary M. Johnson
Julie M. Swisshelm
Daniel V. Pryor
John P. Ziebarth

Institute for Computational Studies
PO Box 1852
Fort Collins, Colorado 80522
U.S.A.

Accepted for Presentation at the
Tenth International Conference on Numerical
Methods in Fluid Dynamics
Beijing, China
23-27 June 1986

SUMMARY

An efficient algorithm designed to be used for Navier-Stokes simulations of complex flows over complete configurations is presented and evaluated. The algorithm incorporates a number of elements, including an explicit three-dimensional flow solver, embedded mesh refinements, a model equation hierarchy ranging from the Euler equations through the full Navier-Stokes equations, multiple-grid convergence acceleration and extensive vectorization and multitasking for efficient execution on parallel-processing supercomputers. Results are presented for a problem representative of turbomachinery applications. Based on the performance data available at this writing, it is expected that the final version of this paper will report overall speedups ranging as high as 100.

INTRODUCTION

Numerical flow simulation is becoming indispensible in aerodynamic design. Because of the large economic benefit resultant from the intelligent use of computational aerodynamics, significant resources are being committed to its application to component design and integration. A recent survey [1] of the work that led to the Boeing 757 and 767 aircraft reveals that computational methods contributed to the design of nearly every aerodynamic component.

The successes attained thus far have raised expectations and established the numerical simulation of complex flows over complete configurations as the next objective. Such capability will permit the design of aerodynamic devices as entire entities, rather than as individual components with their interactions taken into account only through *a-posteriori* modification and integration techniques, as is current practice. This latest goal of computational aerodynamics is being actively pursued. In fact, a first attempt at solving the Navier-Stokes equations for the flowfield around a complete aircraft has already been reported [2].

It is generally recognized that a comprehensive approach to the simulation of flows involving both complex geometries and complex physics will require powerful advanced-architecture supercomputers with very large memories. Machines capable of producing solutions to Reynolds-averaged Navier-Stokes flows over complex geometries within computing times short enough to be of design interest are expected to be available by the end of this decade [3]. In order to use these parallel-processing supercomputers effectively, algorithms must be adapted to focus the power of multiple processing units on a single flow simulation. Furthermore, the history of computational aerodynamics teaches that the pace of progress in this field is set by the synergism between improved computers and better algorithms. In the past 15 years, improved computers have reduced the cost of computation by a factor of about 100. Over the same period, better algorithms have reduced the cost of computation on a given computer by a factor of almost 1000 [4]. Thus, it is to be expected that the need for faster algorithms will not be diminished by the availability of faster and larger computers.

The purpose of the present work is to contribute to the development of algorithms appropriate for the simulation of complex flows over complete configurations. Such algorithms must be efficient and must map readily onto the architectures of parallel-processing supercomputers. The approach selected enhances the efficiency of a robust and flexible solution procedure by implementing it on a collection of local meshes embedded in a global mesh. Either the Euler, thin-layer Navier-Stokes or full Navier-Stokes equations are solved on each mesh. The choice of model equations is determined by the nature of the flow physics to be resolved on a particular mesh. When the requirement for time accuracy is relaxed, a convergence acceleration procedure is applied simultaneously to all meshes and all model equations. The entire algorithm is explicit and is designed to perform well on computers consisting of multiple processing units, each having vector processing capability. Examples of such machines are the Cray X-MP and Cray 2.

Three-dimensional Navier-Stokes simulations using implicit methods have been reported recently by several investigators. These include: Aki and Yamada [5], Deiwert, et al. [6], Flores [7], Fujii and Obayashi [8], Holst, et al. [9], Hung [10], Kordulla [11] and Li [12]. Recent 3-D Navier-Stokes simulations using explicit methods include the work of: Johnson and Swisshelm [13], Mikartarian, et al. [14], Roger, et al. [15] and Shang and Scherr [2]. Interesting 3-D Euler calculations include publications by Koeck and Chattot [16] and Rizzi and Purcell [17]. Some form of three-dimensional zonal gridding or grid embedding has been emphasized in [7], [9] and by Benek, et al. [18]. Interesting work on grid embedding in two dimensions has been carried out by Usab [19] and Eberhardt and Baganoff [20]. Convergence acceleration has been stressed in [7], [13], [16] and [19]. The implementation of Navier-Stokes algorithms on parallel-processing supercomputers has been discussed by Johnson, et al. [21] and Stevens [22].

It thus appears that the time is ripe for the introduction of a solution methodology for complex, three-dimensional viscous flows which embodies the following elements: a robust basic flow solver, embedded meshes, a hierarchy of physical flow models, convergence acceleration and multitasking for efficient execution on parallel-processing supercomputers. Such a methodology is

described in this paper. Computational results are presented for a three-dimensional flow problem related to turbomachinery applications.

EQUATIONS OF MOTION

The nondimensional Reynolds-averaged Navier-Stokes equations may be written in conservation-law form as

$$q_t = -(F_z + G_y + H_z) \tag{1}$$

where, for the full Navier-Stokes equations,

$$F = f - \mathrm{Re}^{-1} p \qquad G = g - \mathrm{Re}^{-1} r \qquad H = h - \mathrm{Re}^{-1} s$$

while, for their thin-layer version,

$$F = f \qquad G = g \qquad H = h - \mathrm{Re}^{-1} d$$

and, for the Euler equations,

$$F = f \qquad G = g \qquad H = h$$

where:

$$q = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ E \end{bmatrix} \quad f = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ \rho u w \\ (E+p)u \end{bmatrix} \quad g = \begin{bmatrix} \rho v \\ \rho u v \\ \rho v^2 + p \\ \rho w v \\ (E+p)v \end{bmatrix} \quad h = \begin{bmatrix} \rho w \\ \rho u w \\ \rho v w \\ \rho w^2 + p \\ (E+p)w \end{bmatrix}$$

$$p = \begin{bmatrix} 0 \\ \tau_{zz} \\ \tau_{yz} \\ \tau_{zz} \\ \beta_z \end{bmatrix} \quad r = \begin{bmatrix} 0 \\ \tau_{zy} \\ \tau_{yy} \\ \tau_{zy} \\ \beta_y \end{bmatrix} \quad s = \begin{bmatrix} 0 \\ \tau_{zz} \\ \tau_{yz} \\ \tau_{zz} \\ \beta_z \end{bmatrix} \quad d = \begin{bmatrix} 0 \\ \mu u_z \\ \mu v_z \\ (\lambda + 2\mu) w_z \\ \gamma \kappa Pr^{-1} e_z + (\lambda + 2\mu) w w_z \end{bmatrix}$$

$$\tau_{xx} = \lambda(u_x + v_y + w_z) + 2\mu u_x \qquad \beta_x = \gamma\kappa Pr^{-1} e_x + u\tau_{xx} + v\tau_{xy} + w\tau_{xz}$$

$$\tau_{yy} = \lambda(u_x + v_y + w_z) + 2\mu v_y \qquad \beta_y = \gamma\kappa Pr^{-1} e_y + u\tau_{yx} + v\tau_{yy} + w\tau_{yz}$$

$$\tau_{zz} = \lambda(u_x + v_y + w_z) + 2\mu w_z \qquad \beta_z = \gamma\kappa Pr^{-1} e_z + u\tau_{zx} + v\tau_{zy} + w\tau_{zz}$$

$$\tau_{xy} = \tau_{yx} = \mu(u_y + v_x)$$

$$\tau_{xz} = \tau_{zx} = \mu(u_z + w_x)$$

$$\tau_{yz} = \tau_{zy} = \mu(v_z + w_y)$$

Here ρ, u, v, w, p and E are respectively density, velocity components in the x-, y- and z-directions, pressure and total energy per unit volume. This final quantity may be expressed as

$$E = \rho\left(e + \frac{1}{2}(u^2 + v^2 + w^2)\right)$$

where the specific internal energy, e, is related to the pressure and density by the simple law of a calorically-perfect gas

$$p = (\gamma - 1)\rho e$$

with γ denoting the ratio of specific heats. The coefficient of thermal conductivity, κ, and the viscosity coefficients, λ and μ, are assumed to be functions only of temperature. Furthermore, by invoking Stokes' assumption of zero bulk viscosity, λ may be expressed in terms of the dynamic viscosity μ as

$$\lambda = -\frac{2}{3}\mu$$

Re and Pr denote the Reynolds and Prandtl numbers, respectively.

Although, for simplicity, the equations of motion are presented here written in Cartesian coordinates, it is well known that their strong conservation law form may be maintained under an arbitrary time-dependent transformation of coordinates. Explicit detail concerning the generalized-coordinate version of these equations, which is employed in the computations to be discussed subsequently is available in [23].

Note that, in practice, the thin-layer assumption is implemented by using a body-fitted coordinate system and neglecting the viscous terms in the coordinate directions along the body. For Cartesian coordinates, with x and y representing the body-conforming coordinates, the thin-layer version of the Navier-Stokes equations is as given above.

The effects of turbulence are simulated by means of a two-layer algebraic eddy viscosity model. In the stress terms of the Navier-Stokes equations, the coefficient of dynamic viscosity, μ, is replaced by $\mu + \mu_t$, where μ_t is the coefficient of eddy viscosity. Similarly, in the heat flux terms, the coefficient of thermal conductivity, κ, is replaced by $\kappa + c_p \mu_t / Pr_t$, where Pr_t is the turbulent Prandtl number. The eddy viscosity is determined by the method of Baldwin and Lomax [24].

SOLUTION METHODOLOGY

In order to minimize the cost of simulating complex, three-dimensional viscous flow over complete configurations, the following strategy has been developed:

a. Use a robust and flexible explicit flow solver capable of simulating either steady or unsteady flow with the Euler, thin-layer Navier-Stokes or full Navier-Stokes equations.

b. Distribute grid points optimally by making use of locally-embedded grid refinements.

c. Make use of a flow simulation hierarchy ranging from the Euler equations

through the full Navier-Stokes equations in order to minimize the computational work per grid point.

d. Accelerate the convergence of steady flow simulations by means of an explicit multiple-grid technique which may be applied, without modification, to the entire hierarchy of model equations.

e. Take advantage of the explicit nature of the algorithm by mapping it onto a supercomputer architecture consisting of multiple vector-processing CPU's and thus enhance its performance by means of both vectorization and multitasking.

Further detail concerning this strategy is provided below.

Basic Flow Solver

The integration scheme used here is the two-step Lax-Wendroff method due to MacCormack [25]. The forward predictor - backward corrector version of this method may be written as

$$\Delta q_{i,j,k} = -\frac{\Delta t}{\Delta x}\left(F^n_{i+1,j,k} - F^n_{i,j,k}\right) - \frac{\Delta t}{\Delta y}\left(G^n_{i,j+1,k} - G^n_{i,j,k}\right)$$

$$-\frac{\Delta t}{\Delta z}\left(H^n_{i,j,k+1} - H^n_{i,j,k}\right)$$

$$\delta q_{i,j,k} = -\frac{\Delta t}{2\Delta x}\left[(F^n_{i+1,j,k} - F^n_{i,j,k}) + (F'_{i,j,k} - F'_{i-1,j,k})\right] \tag{2}$$

$$-\frac{\Delta t}{2\Delta y}\left[(G^n_{i,j+1,k} - G^n_{i,j,k}) + (G'_{i,j,k} - G'_{i,j-1,k})\right]$$

$$-\frac{\Delta t}{2\Delta z}\left[(H^n_{i,j,k+1} - H^n_{i,j,k}) + (H'_{i,j,k} - H'_{i,j,k-1})\right]$$

where:

$$\delta q_{i,j,k} = \Big(q(t+\Delta t) - q(t)\Big)_{i,j,k}$$

$$q'_{i,j,k} = q^n_{i,j,k} + \Delta q_{i,j,k}$$

$$F'_{i,j,k} = F(q'_{i,j,k}) \qquad G'_{i,j,k} = G(q'_{i,j,k}) \qquad H'_{i,j,k} = H(q'_{i,j,k})$$

First derivatives in the viscous terms are backward differenced in the predictor and forward differenced in the corrector.

This version of MacCormack's scheme is used here for convenience. Any of its many variants could also be used, as could any other one- or two-step Lax-Wendroff scheme [26]. In fact, the class of fine-grid methods with which the convergence acceleration technique described below may be applied appears to be quite large, including schemes not of Lax-Wendroff type [27].

The advantages of MacCormack's method, in the present context, are its explicit nature, simplicity and low operations count. A disadvantage is its conditional stability and the severe time-step size limitation which this imposes for viscous flows, in particular. The ill effects of conditional stability are mitigated through the use of embedded grid refinements and convergence acceleration.

Embedded Meshes

The embedded-mesh technique developed for the present application is a generalization of that employed in [19] to obtain two-dimensional Euler solutions. The computational domain is divided into regions requiring grids of differing fineness and the resolution of different flow physics. At present, for simplicity, this partitioning is done *a-priori*. However, solution-adaptive gridding based on this technique is possible. Figures 1 through 5 show typical locations for the mesh regions employed in the computations described subsequently in this paper. Note that, where mesh lines are illustrated, their spacing is much coarser than that employed in the computations. This has been done for clarity of illustration. Figure 1 shows mesh 3, in white. This mesh covers the entire computational domain. Figure 2 overlays, in yellow, the regions collectively referred to as mesh 2. Figure 3 shows, in cyan, the mesh-1 regions projected onto mesh 3. In Figure 4, meshes 1 through 3 are assembled into a

single illustration. Figure 5 simply adds wing surface grid lines to this assemblage. The mesh-1 regions contain the finest grids. Mesh-2 regions are coarser by a factor of 2 in each dimension. Mesh 3 is coarser again by a factor of 2 in each dimension. From this example, it is easy to see that quite general collections of embedded meshes may be constructed in this manner. The embedded meshes are not disjoint. Rather, given a mesh labelled m, all coarser meshes from m+1 through the coarsest mesh used in the computation underlie it. This property, together with the coarsening factor of 2, facilitate the use of the multiple-grid convergence technique described below.

Points on a boundary between two mesh regions belong to the coarser region. Where mesh regions reach the boundaries of the computational domain, boundary conditions are applied consistent with the finest mesh reaching the boundary segment in question. The flowfield updating begins with mesh 1. After one timestep on mesh 1, mesh 2 is updated exterior to mesh 1 while convergence acceleration is applied at the mesh-2 points interior to mesh 1. Next, mesh 3 is updated exterior to mesh 2 while convergence acceleration is applied at the mesh-3 points interior to mesh 2. Updating proceeds in this fashion until the global mesh has been advanced by one timestep. Then the updating cycle is completed by applying convergence acceleration to coarsenings of the global grid. This cycle is repeated until the desired measure of convergence is satisfied.

Simulation Hierarchy

In order to minimize the computational work to obtain a flowfield solution of specified accuracy and physical resolution, different approximations to the Navier-Stokes equations are used in different mesh regions. In the example illustrated in Figures 1 through 5, the full Navier-Stokes equations are solved on mesh 1, the thin-layer Navier-Stokes equations are solved on mesh 2 and the Euler equations are solved on mesh 3. In this way the physics contained in the model equations and the resolving power of the various meshes are matched to each other.

Convergence Acceleration

Given the fine-grid corrections, δq, we wish to use successively coarser grids to propagate this information throughout the computational domain, thus accelerating convergence to the steady state while maintaining the accuracy determined by the fine-grid discretization. Given a basic fine grid with the number of points in each direction expressible as $n(2^p)+1$ for p and n integers such that $p \geqslant 0$ and $n \geqslant 2$, where p is the number of grid coarsenings and n is the number of coarsest-grid intervals, let successively coarser grids be defined by successive deletion of every other point in each coordinate direction.

Although it is quite probable that a large variety of coarse-grid acceleration schemes may be constructed, we limit our attention here to those explicit schemes based on Lax-Wendroff methods. Such an acceleration scheme may be expressed as

$$\delta q_{coarse} = \Delta t \; q_t + \frac{\Delta t^2}{2} q_{tt}$$

By introducing Eqn.(1), this may be rewritten as

$$\delta q_{coarse} = -\Delta t(F_x + G_y + H_z) - \frac{\Delta t^2}{2}(F_x + G_y + H_z)_t$$

Observing that

$$\Delta q = -\Delta t(F_x + G_y + H_z)$$

we obtain

$$\delta q_{coarse} = \Delta q - \frac{\Delta t^2}{2}(F_x + G_y + H_z)_t \tag{3}$$

Various acceleration schemes may now be derived, according to the way in which the second term on the right-hand side of Eqn.(3) is treated.

If we let

$$-(F_x + G_y + H_z)_t = [A(F_x + G_y + H_z)]_x + [B(F_x + G_y + H_z)]_y + [C(F_x + G_y + H_z)]_z$$

where A, B and C are the Jacobian matrices

$$A = \partial F/\partial q \ , \quad B = \partial G/\partial q \quad \text{and} \quad C = \partial H/\partial q$$

we obtain the class of Jacobian-based acceleration schemes, of which the method due to Ni [28] is a member.

If, on the other hand, we let

$$(F_x + G_y + H_z)_t \approx \frac{1}{\Delta t}\Big[(F_x + G_y + H_z)^{n+1} - (F_x + G_y + H_z)^n \Big]$$

where

$$F^n = F(q^n) \ , \qquad G^n = G(q^n) \ , \qquad H^n = H(q^n)$$

$$F^{n+1} = F(q^n + \Delta q) \ , \quad G^{n+1} = G(q^n + \Delta q) \ , \quad H^{n+1} = H(q^n + \Delta q)$$

we obtain the class of flux-based schemes introduced in [29].

In both classes of schemes, Δq is approximated by a restriction of the fine-grid value of δq and second-order accurate spatial differencing is used. For example, a simple Jacobian-based acceleration scheme may be written as

$$\delta q_{i,j,k} = \frac{1}{8} \sum_{l=\pm 1} \sum_{m=\pm 1} \sum_{n=\pm 1} \left[\left(I - l \frac{\Delta t}{\Delta x} A - m \frac{\Delta t}{\Delta y} B - n \frac{\Delta t}{\Delta z} C \right) \Delta q \right]_{i+l,j+m,k+n}$$

This scheme is contrasted in Fig. 6 with its one-step Lax-Wendroff analog, written on the fine grid. That one-step scheme may be written as

$$\delta q_{i,j,k} = \frac{1}{8} \sum_{l=\pm 1} \sum_{m=\pm 1} \sum_{n=\pm 1} \left[\left(I - l \frac{\Delta t}{\Delta x} A - m \frac{\Delta t}{\Delta y} B - n \frac{\Delta t}{\Delta z} C \right) \Delta q \right]_{i+\frac{l}{2},\, j+\frac{m}{2},\, k+\frac{n}{2}}$$

where Δq is not approximated as a restriction of some δq, as in the coarse-grid scheme, but is rather computed as

$$\Delta q_{i+\frac{1}{2},\, j+\frac{1}{2},\, k+\frac{1}{2}} =$$

$$- \frac{\Delta t}{4\Delta x} \Big[\Big(F_{i+1,j,k} + F_{i+1,j+1,k} + F_{i+1,j,k+1} + F_{i+1,j+1,k+1} \Big)$$

$$- \Big(F_{i,j,k} + F_{i,j+1,k} + F_{i,j,k+1} + F_{i,j+1,k+1} \Big) \Big]$$

$$- \frac{\Delta t}{4\Delta y} \Big[\Big(G_{i,j+1,k} + G_{i+1,j+1,k} + G_{i,j+1,k+1} + G_{i+1,j+1,k+1} \Big)$$

$$- \Big(G_{i,j,k} + G_{i+1,j,k} + G_{i,j,k+1} + G_{i+1,j,k+1} \Big) \Big]$$

$$- \frac{\Delta t}{4\Delta z} \Big[\Big(H_{i,j,k+1} + H_{i+1,j,k+1} + H_{i,j+1,k+1} + H_{i+1,j+1,k+1} \Big)$$

$$- \Big(H_{i,j,k} + H_{i+1,j,k} + H_{i,j+1,k} + H_{i+1,j+1,k} \Big) \Big] .$$

All results reported in this paper use flux-based convergence acceleration.

In the sequential grid updating algorithm, illustrated in Fig. 7a, the solution is advanced over one multiple-grid cycle as follows. First a fine-grid correction, δq_1, is computed. Then δq_1 is restricted to the next-coarser grid, where δq_2 is

computed. The δq_2 correction is both restricted to grid 3 and prolonged to grid 1, where it provides an additional update to the fine-grid solution. On grids 3 through N-1 the procedure is analogous to that on grid 2. When δq_N has been computed and prolonged to grid 1 to provide the Nth update to the fine-grid solution, the next multiple-grid cycle is ready to begin.

Observe that when the components of the sequential grid updating algorithm (namely, the fine- and coarse-grid schemes) are both explicit, it is particularly easy to vectorize. However, the effectiveness of vectorizing the coarse-grid scheme is limited by the progressively shorter vectors which may be constructed on the successively coarser grids. Such an explicit sequential algorithm may also be run on a Multiple Instruction-Multiple Data (MIMD) machine by splitting each grid, in turn, across the total number of processors available. An implicit sequential grid updating algorithm would probably not vectorize as well and also require additional redesign to run on a parallel processor.

The parallel coarse-grid algorithm, illustrated in Fig. 7b, removes the dependence of grids 3 through N upon their immediate predecessors. In particular, δq_1 is now restricted to each of grids 2 through N. All of these coarse grids may then be updated simultaneously and independently of each other. This allows the mesh points on grids 2 through N to be assembled into one vector in order to improve performance on a Single Instruction-Multiple Data (SIMD) computer. Alternatively, the coarse grids could each be updated simultaneously on separate processors of an MIMD machine. This would be attractive, for example, if the coarse-grid scheme were implicit.

A further possibility is the fully parallel algorithm, illustrated in Fig. 7c. Here δq_1 from the previous cycle is restricted to each of the coarse grids. This makes all of the grids 1 through N independent of each other and allows their simultaneous update.

Observe that dissipative effects have a local character and their influence need not be taken into account in the construction of acceleration schemes. Rather, it is the convective terms, with their global character, which are the key element in coarse-grid propagation. Hence, acceleration schemes for viscous

flow computations may be formulated on the basis of the inviscid equations of motion. Such a scheme leads to a convergence acceleration procedure which is independent of the nature of the dissipative terms retained in the viscous model equations. That is to say: a scheme based on the Euler equations may be employed, without modification, to accelerate the convergence of viscous flow computations based on the Navier-Stokes equations, the thin-layer equations, or any other viscous model equations which contain the full inviscid Euler equations.

Boundary conditions are not updated during convergence acceleration. This has the advantage of decoupling the acceleration scheme from both the physical and numerical nature of these boundary conditions. That is to say: the acceleration scheme always sees a Dirichlet problem. Any numerical damping terms which may be necessary are also omitted during convergence acceleration. This enhances the modularity of the acceleration scheme.

For all of the results to be discussed subsequently, linear interpolation has been used as the prolongation operator. For the sequential grid updating algorithm, injection is used as the restriction operator. The parallel coarse-grid algorithm introduces averaging into the restriction operator for grids 3 through N. The fully parallel algorithm further employs underrelaxation of the fine-grid information restricted from the previous cycle.

Finally, note that, while in this paper multiple-grid convergence acceleration is applied only to steady flow simulations, it appears that the technique may extend to the time-accurate computation of some unsteady flows [30, 31].

Multitasking

When attempting to multitask an algorithm for execution on an MIMD machine, we are concerned with multitasking overhead and algorithm granularity. By granularity we mean the time required to execute a multitaskable segment of the algorithm on a single processor [32]. For a given multitasking overhead, the best speedup is obtained when algorithm granularity is maximal. Large granularity is usually introduced by top-down programming which

exploits global parallelism in the algorithm. Bottom-up programming, on the other hand, exploits algorithm parallelism at a low level by making many partitionings, each on small code segments, such as DO loops containing independent statements [33].

The sequential multigrid algorithm contains many opportunities for creating small granularity parallelism but relatively few opportunities for the sort of large granularity necessary to produce good multitasking speedup in the face of non-trivial multitasking overhead. This observation, together with the desirability of non-sequential multigrid schemes for reasons of algorithm flexibility, led to the construction of the parallel multigrid algorithms described above. In these algorithms, grids which are independent of one another may be updated simultaneously on separate processors. In fact, such a simple strategy may result in a poor load balance across processors because of the differing amounts of work inherent in updating grids of different coarseness. However, more refined strategies are possible. Grids may, for example, be grouped together into tasks of approximately equal work, or they may be melded into tasks with other large-grained multitaskable code segments in order to equilibrate processor loading. Notice further that, by multitasking large-grained structures, the vectorization potential of code within these structures remains intact.

NUMERICAL SIMULATION

At present, all of the elements discussed in the section on solution methodology have been encoded and computationally verified in either two or three dimensions, or both. These elements are now being assembled into a comprehensive three-dimensional algorithm. Performance evaluation of this algorithm will be completed in March 1986 and reported in the final version of this paper. For this abstract, although the three-dimensional model problem designed to exercise all aspects of the completed algorithm is described, only interim results are discussed. The expected variance between interim and final results is indicated, where possible.

Physical Problem

As the algorithm described in this paper is designed to efficiently simulate complex flows over complete configurations, it should be tested under conditions which fully exercise its capabilities. On the other hand, excessive complexity would serve no useful purpose in the initial testing phases of the algorithm. With these considerations in mind, three-dimensional computations are being carried out for the geometry illustrated in Fig. 8, a rectilinear cascade of finite-span, swept blades mounted between endwalls. The sweep angle ranges from 0 to 26 degrees. The blade thickness to chord ratio ranges from 0.0 to 0.2. The subcritical computations are performed at an isentropic inlet Mach number of 0.5. The Mach number for the supercritical computations is 0.675. In the viscous cases, the Reynolds numbers, based on cascade gap and critical speed, span the approximate range from 8.4×10^3 to 2.0×10^5.

At the upstream domain boundary, total pressure, total temperature and flow angle are specified. At the downstream boundary, the static pressure is fixed. Along inviscid lateral boundaries, the tangency condition is applied, while, along solid walls, the no-slip condition is applied and the temperature specified. Symmetry and periodicity are invoked to limit the size of the computational domain. Uniform flow at the isentropic inlet Mach number is used as an initial state.

Algorithm Implementation

The mesh structure on which the computations are performed is illustrated in Figs. 1 through 5. The full Navier-Stokes equations are solved on mesh 1. The thin-layer Navier-Stokes equations are solved on mesh 2. The Euler equations are solved on mesh 3. Only steady flows are computed and convergence acceleration, as described previously, is applied. The entire algorithm is vectorized and multitasked to run on a four-processor Cray X-MP or Cray 2.

Computational Results

Sample flowfield results obtained using the basic flow solver with

convergence acceleration but without embedded mesh refinements are shown in Figs. 9 through 12. The use of embedded meshes causes no deterioration of these results, assuming that the meshes are positioned properly. The final version of this paper will contain much more elaborate flowfield results, including depiction of corner vortices.

Comparison of the embedded-mesh algorithm with a single-mesh algorithm yields the following general result: the accuracy of the embedded-mesh results is essentially that of a global finest mesh, while the convergence rate is like that of a global coarsest mesh. This observation is consistent with [19]. Thus far, in two-dimensional computations using the Euler and thin-layer Navier-Stokes equations and three mesh regions, embedding speedups as high as 30 in comparison to a single-mesh algorithm have been obtained. Some interim two-dimensional embedding speedups are reported in Table I. Three-dimensional embeddings using Euler, thin-layer and full Navier-Stokes regions should produce substantially larger speedups.

Multiple-grid convergence acceleration applied to three-dimensional cases, in the absence of mesh embedding, has yielded speedups ranging from 2.5 to 4.7 (see Table II). It is expected that there will be some tradeoff between embedding speedup and multigrid speedup in the complete algorithm.

Vectorization of the three-dimensional algorithm without embedded meshes results in speedups ranging from 3.6 to 5.7 (see Table III). This range should remain about the same in the final algorithm.

Using a top-down multitasking approach, the parallel coarse-grid algorithm has been implemented on a four processor Cray X-MP, for two-dimensional cases without mesh embedding. Initially, only the coarse grids were multitasked so that the performance of parallel grids on a multiprocessor could be evaluated. Then the fine-grid computations were partitioned and multitasked, and the resultant code was integrated with the parallelized coarse grids. Load balancing of the entire scheme completed the study of performance resulting from the top-down approach.

Multitasking results are shown in Table IV. The performance measures are based on a comparison of multitasked code segments with their unitasked analogs. The parallel coarse-grid scheme results contained in Table IVa were obtained with a five-grid multigrid sequence length. Observe that an efficiency of nearly 90% has been obtained using two processors, but that efficiency deteriorates to 77% when four processors are used. This deterioration is a result of distributing multigrid structures containing unequal amounts of work across four processors, which creates a less than ideal load balance. Table IVb shows results obtained from multitasking the fine-grid scheme. The fine-grid tasks are fairly evenly balanced, and this code segment performs well on both two and four processors. Processor utilization of 90% or better is achieved. When the four-processor case is recomputed using the low-overhead version of multitasking known as microtasking [34], efficiency is improved to 95%. The fully multitasked multigrid algorithm performance is shown in Table IVc. On two processors, 94% efficiency is obtained, while on four processors, with speedup by a factor of 3.3 over the unitasked code, efficiency is 83%. Multitasking speedups are expected to improve for the complete three-dimensional code.

Given that the speedups from the various categories described above are generally multiplicative in effect, it is to be conservatively estimated that the completed algorithm to be reported in the final version of this paper will produce overall speedups ranging as high as 100.

CONCLUSIONS

An efficient algorithm designed to be used for Navier-Stokes simulations of complex flows over complete configurations has been presented.

The algorithm makes use of several elements:

A robust explicit basic flow solver

Locally-embedded mesh refinements

A flow simulation hierarchy ranging from the Euler equations through the full Navier-Stokes equations

An explicit multiple-grid convergence acceleration technique

Both vectorization and multitasking for efficient execution on parallel-processing supercomputers

Results are presented for a problem representative of turbomachinery applications. These results validate the algorithm and provide grounds for optimism regarding its future application to more challenging internal and external flows.

Based on the interim performance data available at this writing, it is estimated that the completed algorithm to be reported in the final version of this paper will produce overall speedups ranging as high as 100.

ACKNOWLEDGEMENTS

The research reported here is funded in part by the NASA Ames Research Center (Grant No. NCC-2-344) and the Air Force Office of Scientific Research (Grant No. AFOSR-85-289). Some of the computing time has been contributed by Cray Research, Inc. All of this support is gratefully acknowledged.

REFERENCES

1. Rubbert, P.E.: The Emergence of Advanced Computational Methods in the Aerodynamic Design of Commercial Transport Aircraft. Proceedings of the International Symposium on Computational Fluid Dynamics, Sponsored by the Japan Society of Computational Fluid Dynamics, Tokyo, 1985.

2. Shang, J.S. and Scherr, S.J.: Navier-Stokes Solution of the Flow Field Around a Complete Aircraft. AIAA Paper 85-1509, 1985.

3. Bailey, F.R.: Overview of NASA's Numerical Aerodynamic Simulation Program. Proceedings of the International Symposium on Computational Fluid Dynamics, Sponsored by the Japan Society of Computational Fluid Dynamics, Tokyo, 1985.

4. Peterson, V.L.: Impact of Computers on Aerodnamics Research and Development. Proceedings of the IEEE, Vol. 72, No. 1, January 1984.

5. Aki, T. and Yamada, S.: A Supercomputer Simulation of Three Dimensional Viscous Flow. Proceedings of the International Symposium on Computational Fluid Dynamics, Sponsored by the Japan Society of Computational Fluid Dynamics, Tokyo, 1985.

6. Deiwert, G.S., Rothmund, H.J. and Nakahashi, K.: Simulation of Complex Three-Dimensional Flows. Proceedings of the International Symposium on Computational Fluid Dynamics, Sponsored by the Japan Society of Computational Fluid Dynamics, Tokyo, 1985.

7. Flores, J.: Convergence Acceleration for a Three-Dimensional Euler Navier-Stokes Zonal Approach. AIAA Paper 85-1495, 1985.

8. Fujii, K. and Obayashi, S.: The Development of Efficient Navier-Stokes Codes for Transonic Flow Field Simulations. Proceedings of the International Symposium on Computational Fluid Dynamics, Sponsored by the Japan Society of Computational Fluid Dynamics, Tokyo, 1985.

9. Holst, T.L., Gundy, K.L., Flores, J., Chaderjian, N.M., Kaynak, U. and Thomas, S.D.: Numerical Solution of Transonic Wing Flows Using an Euler/Navier-Stokes Zonal Approach. AIAA Paper 85-1640, 1985.

10. Hung, C.M.: Computation of Three-Dimensional Shock Wave and Boundary-Layer Interactions. Proceedings of the International Symposium on Computational Fluid Dynamics, Sponsored by the Japan Society of Computational Fluid Dynamics, Tokyo, 1985.

11. Kordulla, W.: The Computation of Three-Dimensional Transonic Viscous Flows with Separation. Proceedings of the Ninth International Conference on Numerical Methods in Fluid Dynamics, Lecture Notes in Physics, Vol. 218, Springer, 1985.

12. Li, C.P.: Numerical Procedure for Three-Dimensional Hypersonic Viscous Flow over Aerobrake Configuration. Proceedings of the International Symposium on Computational Fluid Dynamics, Sponsored by the Japan Society of Computational Fluid Dynamics, Tokyo, 1985.

13. Johnson, G.M. and Swisshelm, J.M.: Multiple-Grid Solution of the Three-Dimensional Euler and Navier-Stokes Equations. Proceedings of the Ninth International Conference on Numerical Methods in Fluid Dynamics, Lecture Notes in Physics, Vol. 218, Springer, 1985.

14. Mikatarian, R.R., Rapagnani, N.L. and Hendricks, W.L.: RAVEN: A Computer Model for the Solution of the Time-Dependent, Three Dimensional Navier-Stokes Equations with Nonequilibrium Chemistry. Proceedings of the International Symposium on Computational Fluid Dynamics, Sponsored by the Japan Society of Computational Fluid Dynamics, Tokyo, 1985.

15. Roger, R.P., Thoenes, J. and Robertson, S.J.: Viscous Flow Computations for a Two-Duct Version of the Space Shuttle Main Engine Hot Gas Manifold. Proceedings of the International Symposium on Computational Fluid Dynamics, Sponsored by the Japan Society of Computational Fluid Dynamics, Tokyo, 1985.

16. Koeck, C. and Chattot, J.J.: Computation of Three-Dimensional Vortex Flows Past Wings Using the Euler Equations and a Multiple-Grid Scheme. Proceedings of the Ninth International Conference on Numerical Methods in Fluid Dynamics, Lecture Notes in Physics, Vol. 218, Springer, 1985.

17. Rizzi, A. and Purcell, C.J.: Cyber 205 Exploration of Inviscid Vortex-Stretched Turbulent Shear-Layer Flow. Proceedings of the International Symposium on Computational Fluid Dynamics, Sponsored by the Japan Society of Computational Fluid Dynamics, Tokyo, 1985.

18. Benek, J.A., Buning, P.G. and Steger, J.L.: A 3-D Chimera Grid Embedding Technique. AIAA Paper 85-1523, 1985.

19. Usab, W.J.: Embedded Mesh Solutions of the Euler Equation Using a Multiple-Grid Method. Doctoral Dissertation, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, 1983.

20. Eberhardt, S. and Baganoff, D.: Overset Grids in Compressible Flow. AIAA Paper 85-1524, 1985.

21. Johnson, G.M., Swisshelm, J.M. and Kumar, S.P.: Concurrent-Processing Adaptations of a Multiple-Grid Algorithm. AIAA Paper 85-1508, 1985.

22. Stevens, K.G., Jr.: Efficiency of CFD Algorithms on Multiprocessor Architectures. Proceedings of the International Symposium on Computational Fluid Dynamics, Sponsored by the Japan Society of Computational Fluid Dynamics, Tokyo, 1985.

23. Pulliam, T.H. and Steger, J.L.: Implicit Finite-Difference Simulation of Three-Dimensional Compressible Flow. AIAA J., Vol. 18, No. 2, pp. 159-167, 1980.

24. Baldwin, B.S. and Lomax, H.: Thin-Layer Approximation and Algebraic Model for Separated Turbulent Flows. AIAA Paper 78-257, 1978.

25. MacCormack, R.W.: The Effect of Viscosity in Hypervelocity Impact Cratering. AIAA Paper 69-354, 1969.

26. Johnson, G.M.: Multiple-Grid Acceleration of Lax-Wendroff Algorithms. NASA TM-82843, 1982.

27. Stubbs, R.M.: Multiple-Gridding of the Euler Equations with an Implicit Scheme. AIAA Paper 83-1945, 1983.

28. Ni, R.H.: A Multiple Grid Scheme for Solving the Euler Equations. AIAA Paper 81-1025, 1981.

29. Johnson, G.M.: Flux-Based Acceleration of the Euler Equations. NASA TM-83453, 1983.

30. Stubbs, R.M.: Private Communication, 1983.

31. Jespersen, D.C.: A Time-Accurate Multiple-Grid Algorithm. AIAA Paper 85-1493, 1985.

32. Larson, J.L.: Practical Concerns in Multitasking on the Cray X-MP. Workshop on Using Multiprocessors in Meteorological Models, ECMLOF, Reading, England, 3-6 December 1984.

33. Larson, J.L.: Multitasking on the Cray X-MP-2 Multiprocessor. *Computer*, pp. 62-69, 1984.

34. Booth, M.: Microtasking - A Brief Design Summary. Cray Research, Inc. Internal Memorandum, 1985.

TABLES

Table I. Interim Two-Dimensional Embedding Speedups

Inviscid Subcritical	16.4
Inviscid Supercritical	6.1
Turbulent Viscous	30.2

Table II. Interim Three-Dimensional Multigrid Speedups

Inviscid Subcritical	4.7
Inviscid Supercritical	2.5
Turbulent Viscous	4.4

Table III. Interim Three-Dimensional Vectorization Speedups

	Scalar	Automatic Vectorization	Explicit Vectorization
Inviscid Subcritical	1.0	4.2	5.7
Inviscid Supercritical	1.0	3.1	4.9
Turbulent Viscous	1.0	2.6	3.6

Table IVa. Interim Two-Dimensional Multitasked Coarse-Grid Performance

Machine	2 Processors		4 Processors	
	Speedup	Efficiency	Speedup	Efficiency
Cray X-MP	1.78	0.89	3.06	0.77

Table IVb. Interim Two-Dimensional Multitasked Fine-Grid Performance

Machine	2 Processors		4 Processors	
	Speedup	Efficiency	Speedup	Efficiency
Cray X-MP	1.91	0.96	3.58	0.90
Cray X-MP with microtasking	1.93	0.97	3.78	0.95

Table IVc. Interim Two-Dimensional Multitasked Complete Scheme Performance

Machine	2 Processors		4 Processors	
	Speedup	Efficiency	Speedup	Efficiency
Cray X-MP	1.87	0.94	3.30	0.83

ORIGINAL PAGE IS
OF POOR QUALITY

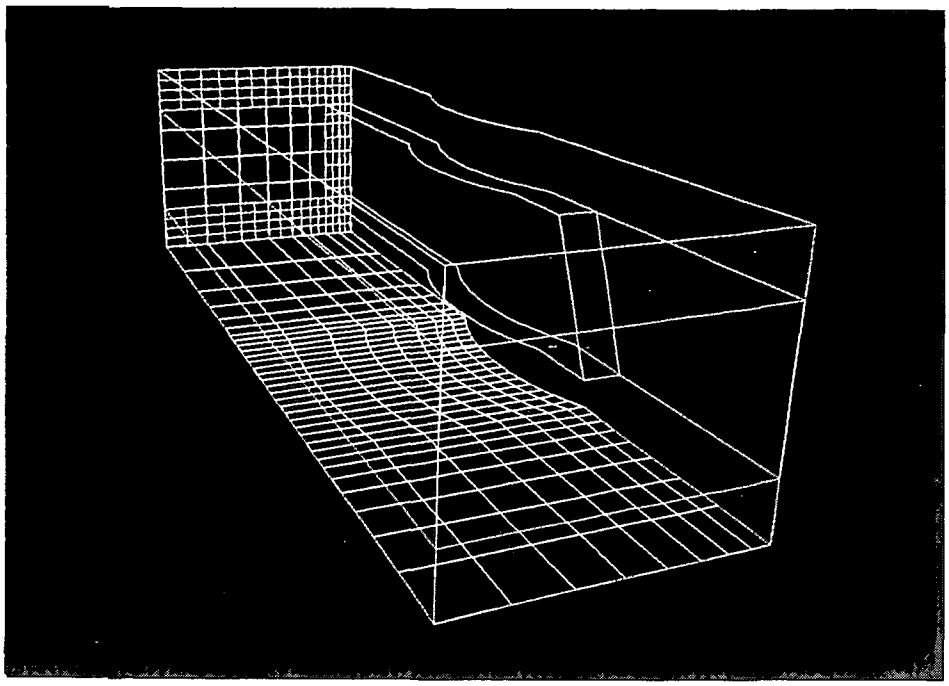


FIGURE 2. Mesh 2, in Yellow, Overlain on Mesh 3

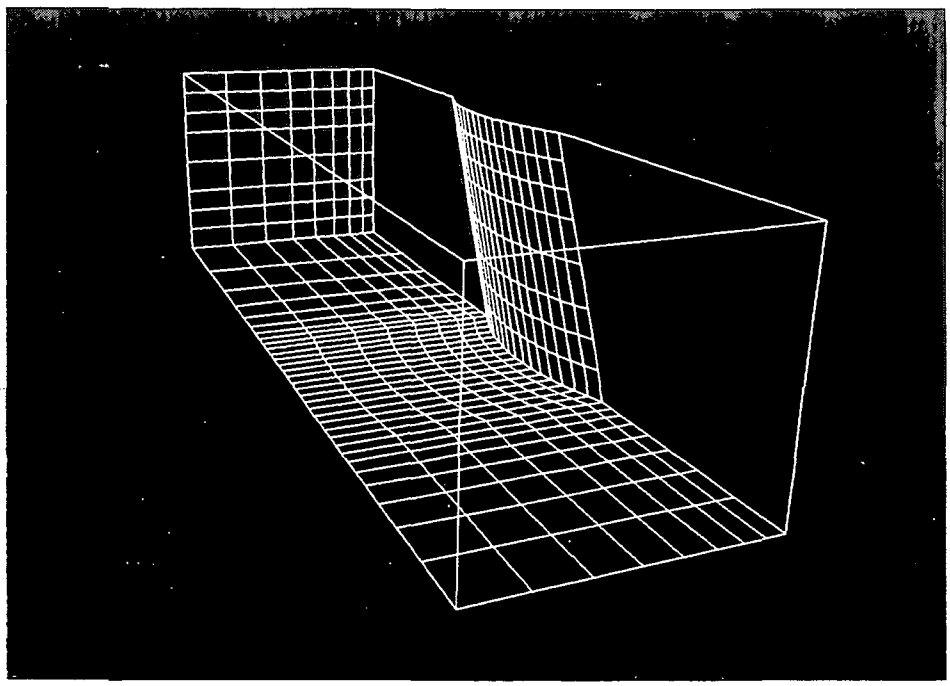


FIGURE 1. Mesh 3

ORIGINAL PAGE IS
OF POOR QUALITY

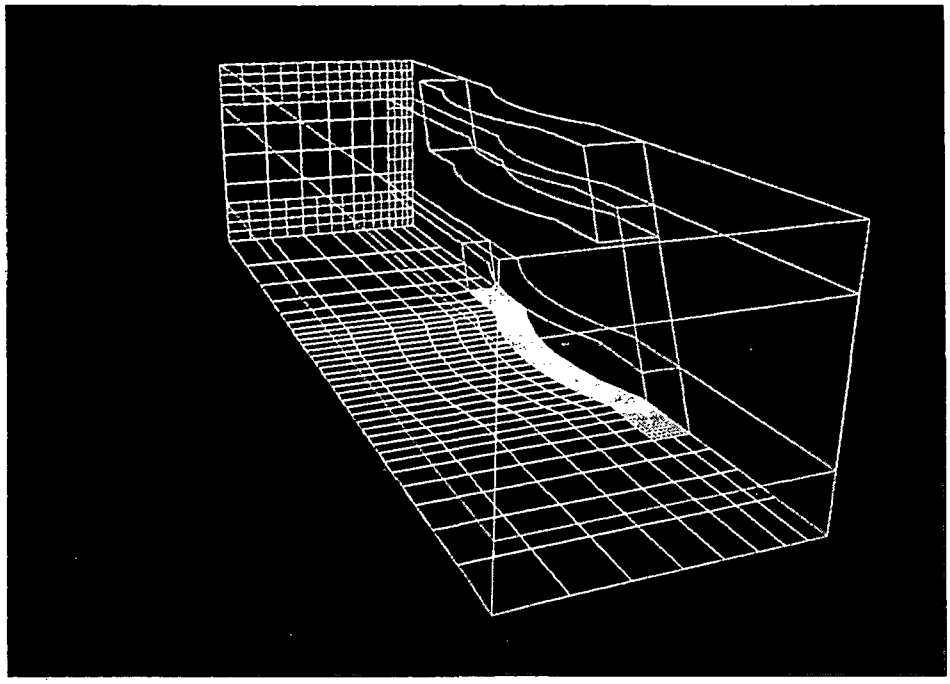


FIGURE 4. Meshes 1 through 3

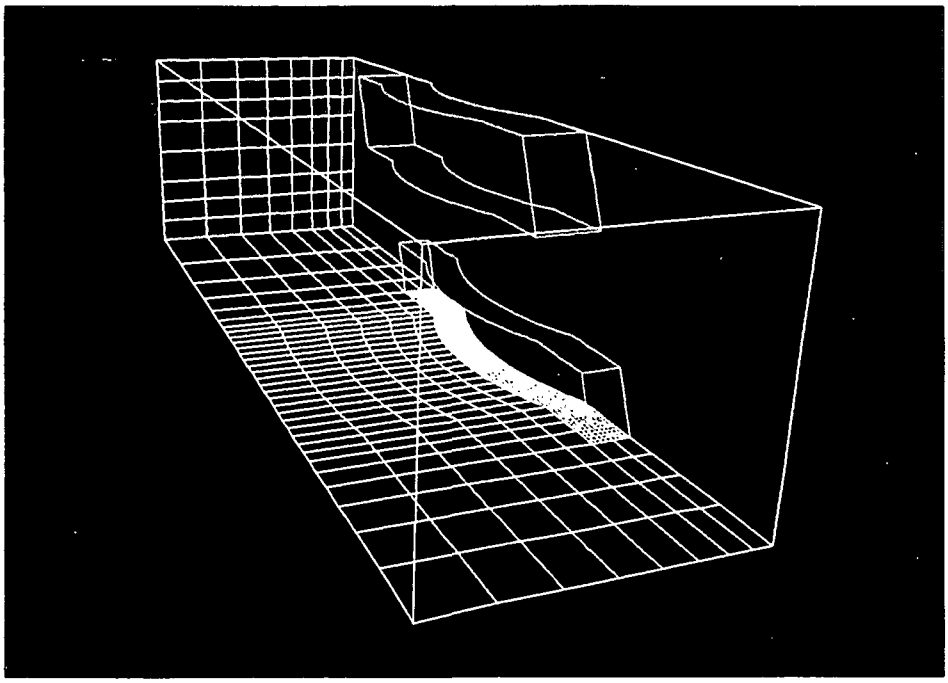


FIGURE 3. Mesh 1, in Cyan, Overlain on Mesh 3

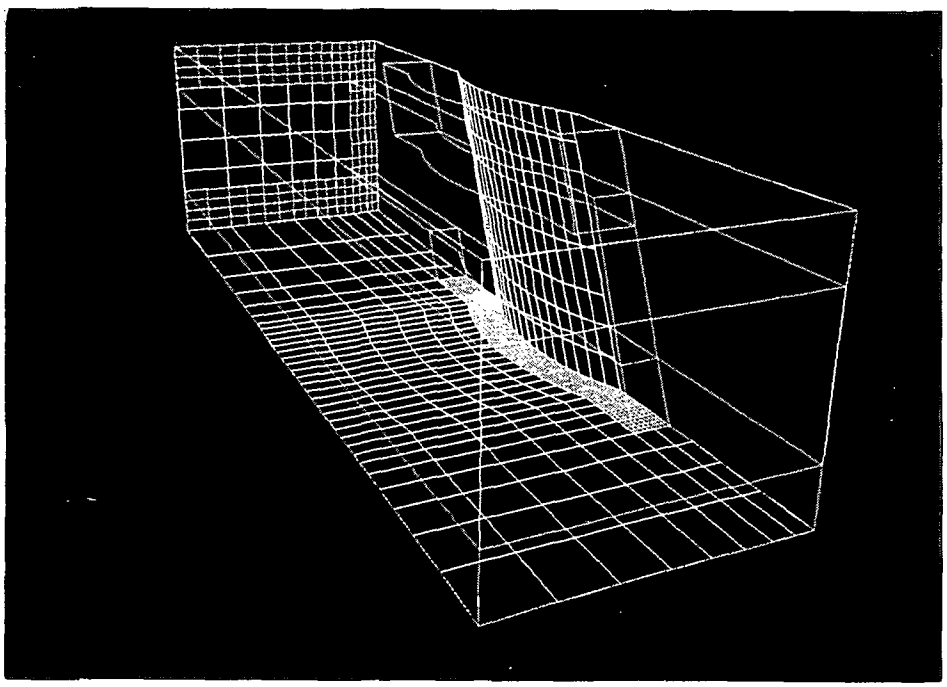
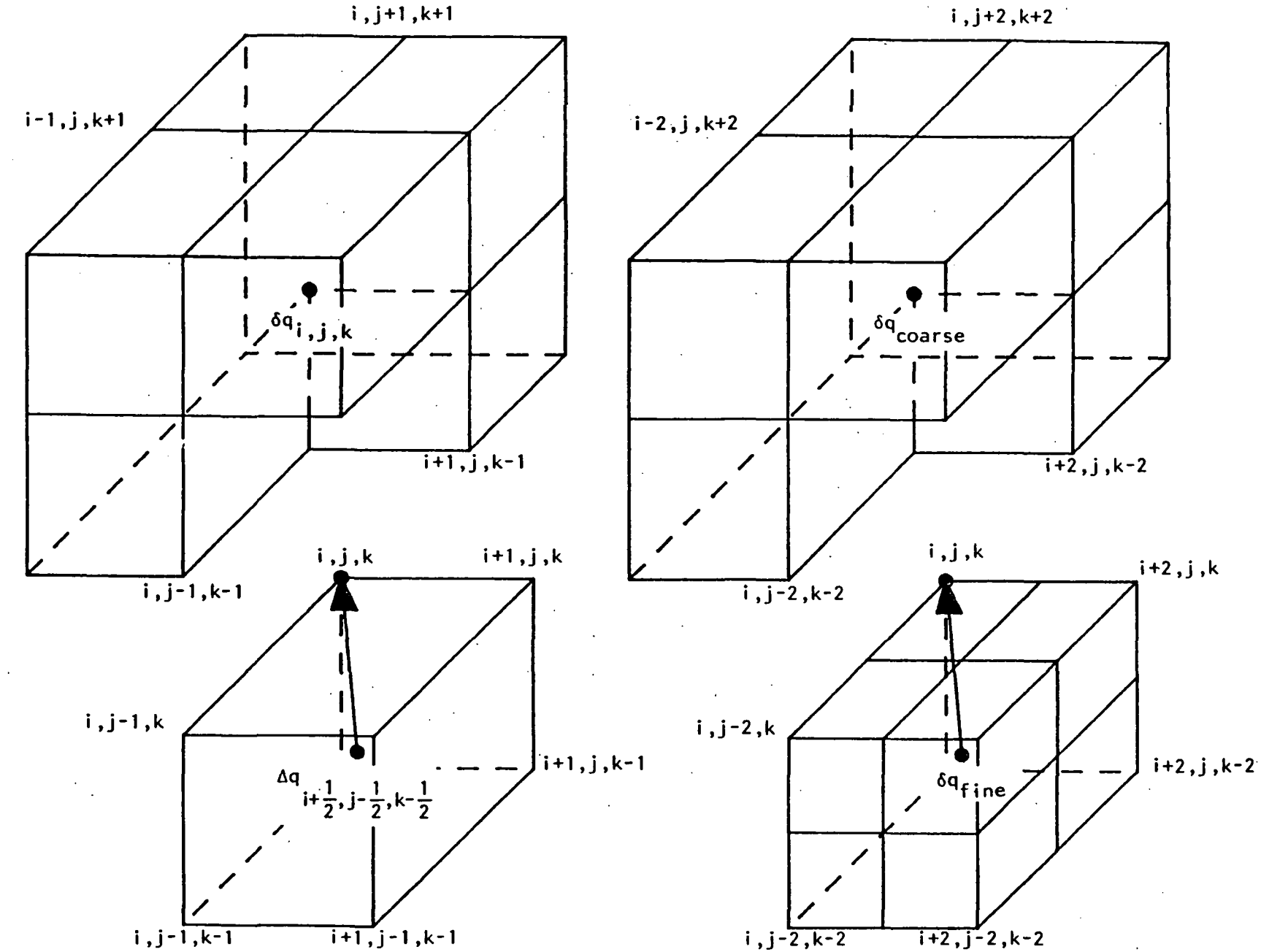


FIGURE 5. Meshes 1 through 3 with Wing Surface Grid Lines Shown



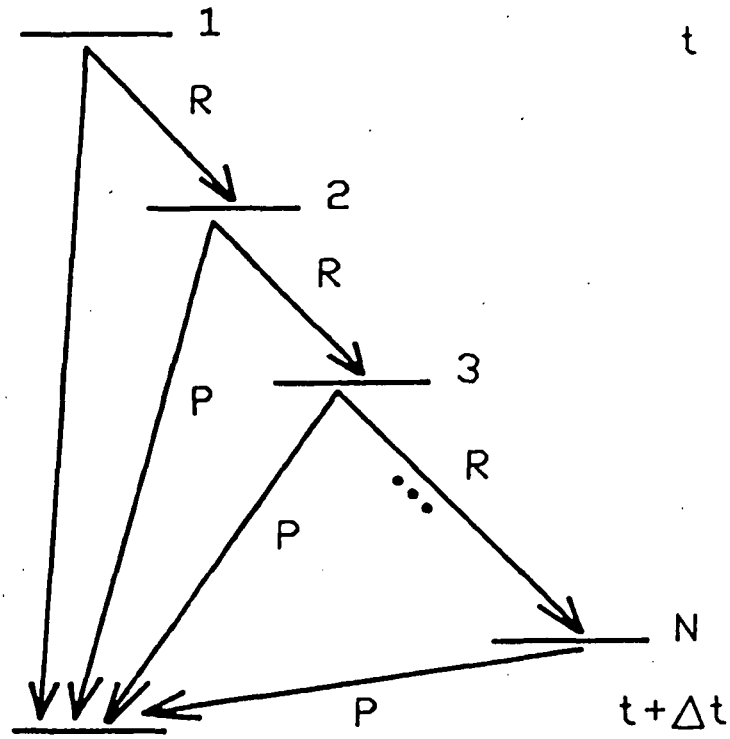
One-Step Lax-Wendroff Scheme on Fine Grid

Coarse-Grid Scheme

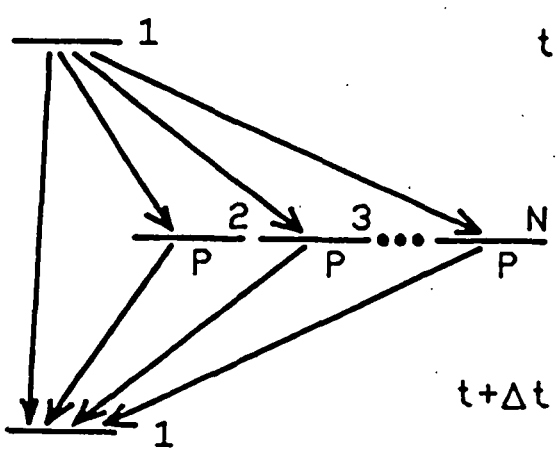
FIGURE 6. Comparison of Fine- and Coarse-Grid Schemes

R - Restriction of δq as Coarse-Grid Δq

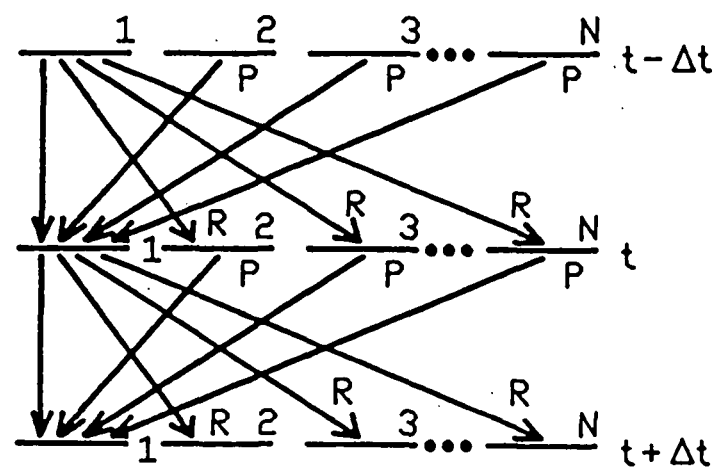
P - Prolongation of δq to Fine Grid



a. Sequential Algorithm



b. Parallel Coarse Grids



c. Fully Parallel Scheme

FIGURE 7. Multiple-Grid Algorithm Information Flow

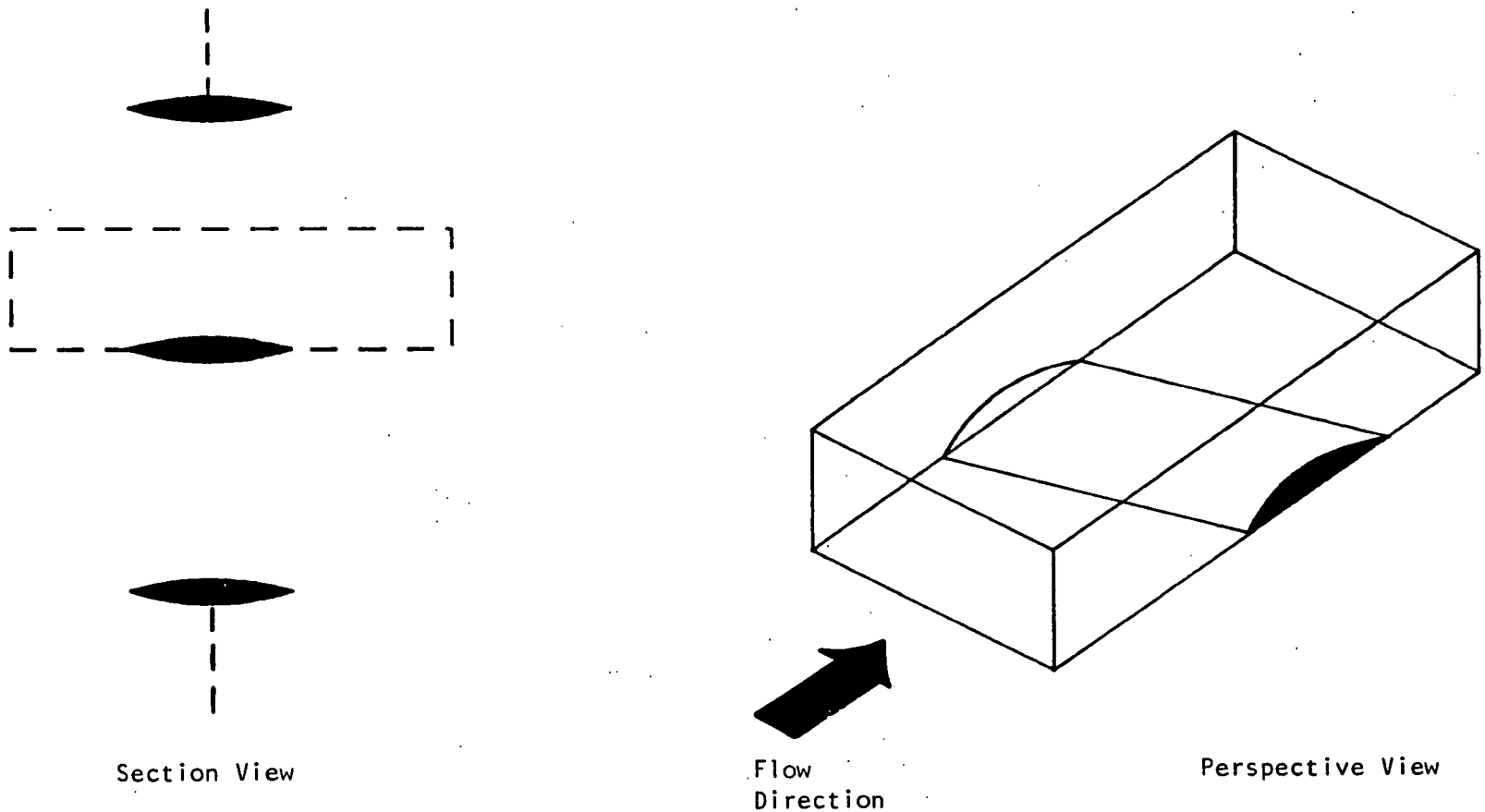


FIGURE 8. Computational Domain for Three-Dimensional Cascade

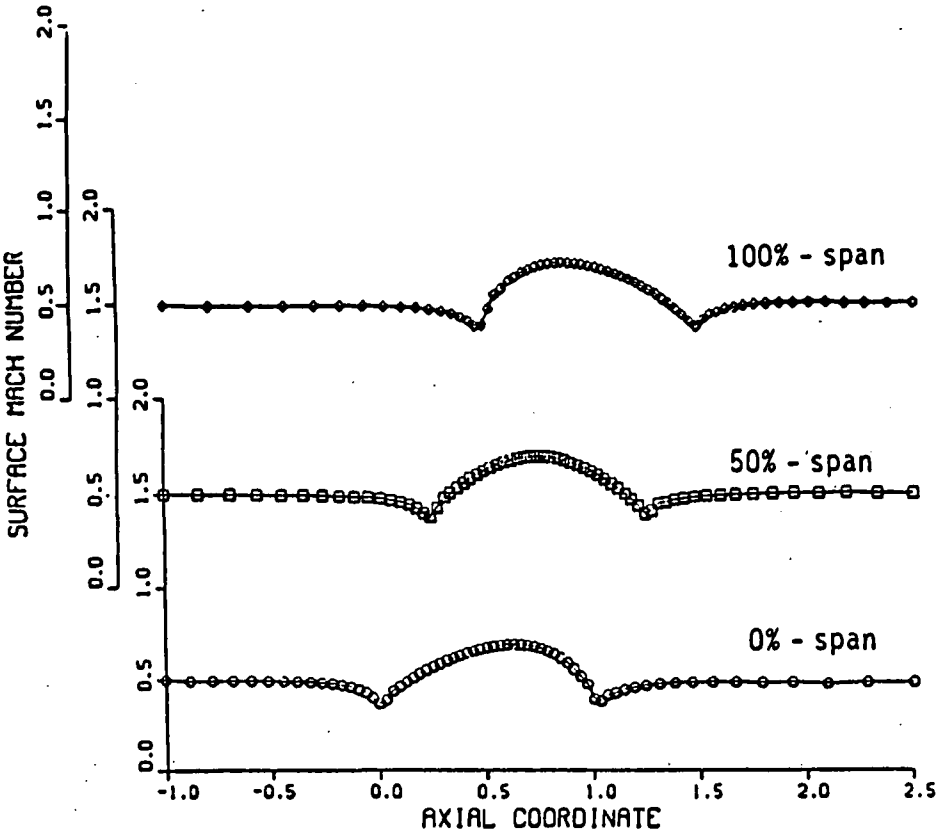


FIGURE 9. Subcritical Inviscid Flow, 26 Deg. Sweep

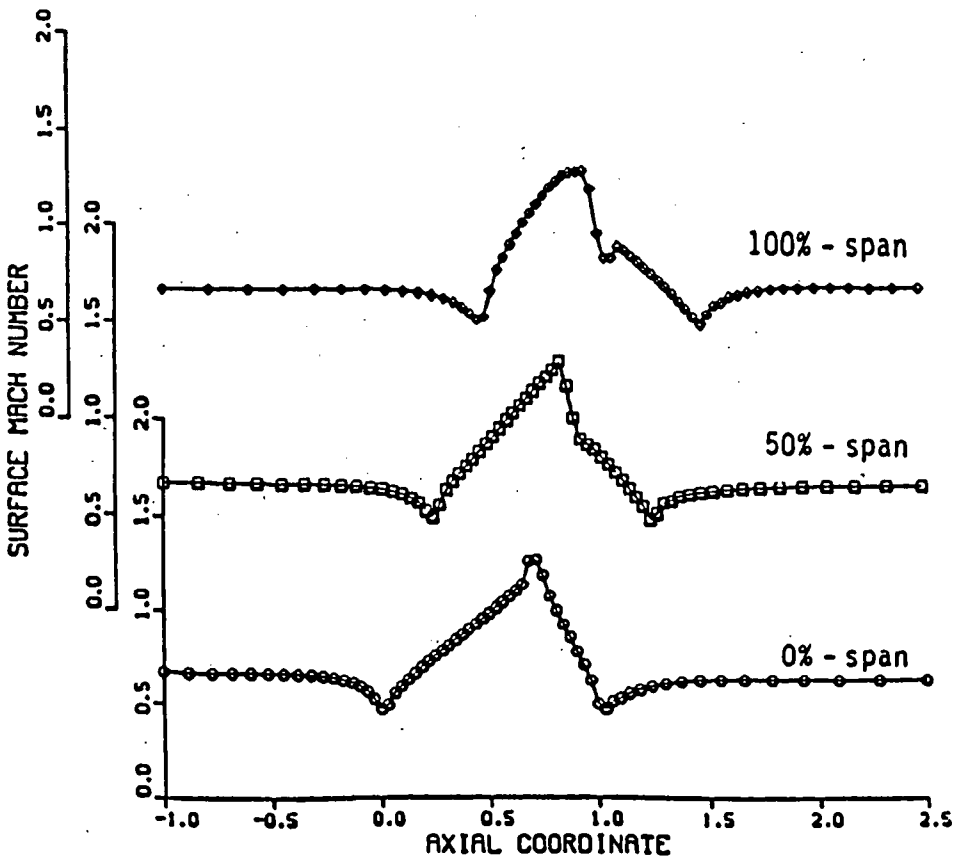


FIGURE 10. Supercritical Inviscid Flow, 26 Deg. Sweep

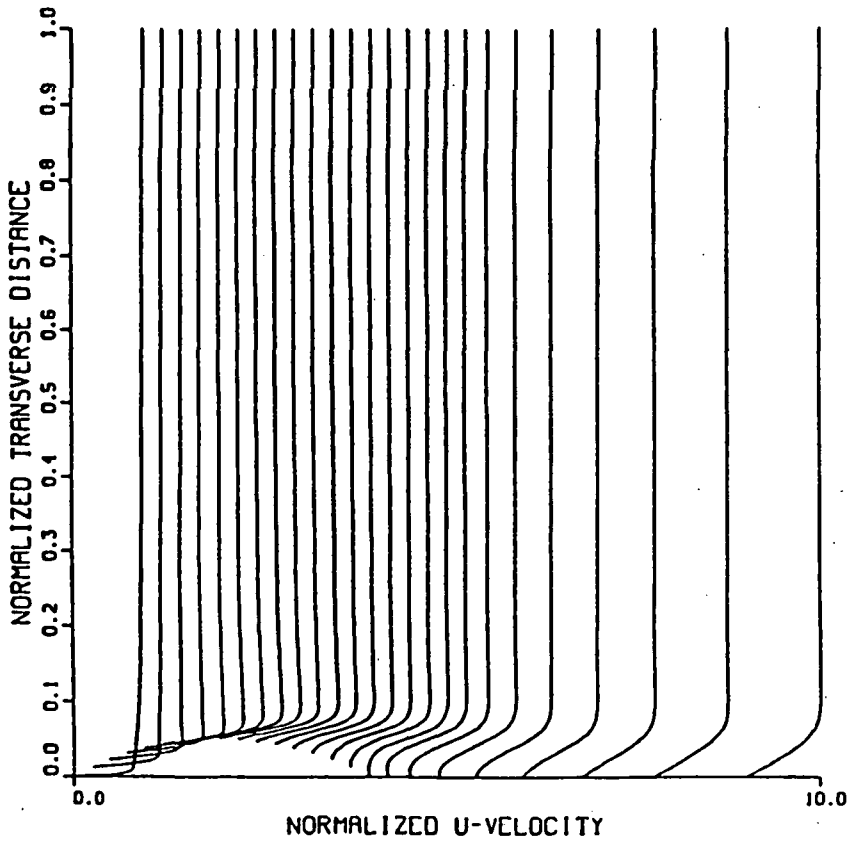


FIGURE 11. Laminar Flow, Re = 3.4×10^4, 26 Deg. Sweep, 50% Span

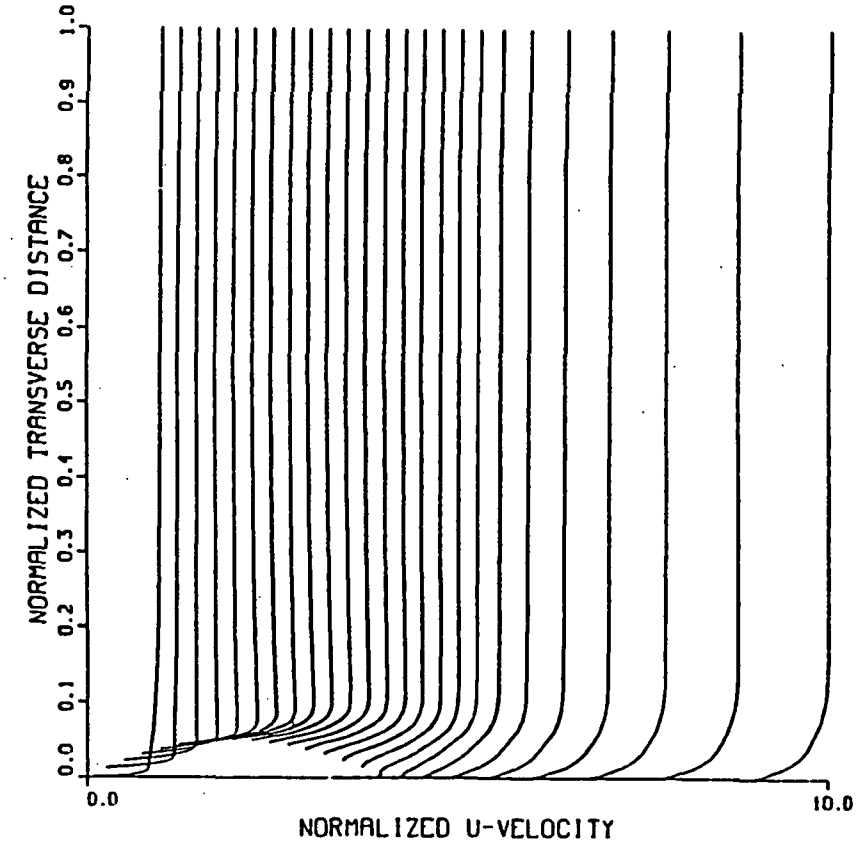


FIGURE 12. Turbulent Flow, Re = 3.4×10^4, 26 Deg. Sweep, 50% Span