

# **SANDIA REPORT**

SAND98-0701 • UC-405

Unlimited Release

Printed April 1998

## **A User's Guide for BREAKUP: A Computer Code for Parallelizing the Overset Grid Approach**

Daniel W. Barnette

Prepared by

Sandia National Laboratories

Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,  
a Lockheed Martin Company, for the United States Department of  
Energy under Contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.



**Sandia National Laboratories**

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

**NOTICE:** This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from  
Office of Scientific and Technical Information  
P.O. Box 62  
Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from  
National Technical Information Service  
U.S. Department of Commerce  
5285 Port Royal Rd  
Springfield, VA 22161

NTIS price codes  
Printed copy: A04  
Microfiche copy: A01



SAND98-0701  
Unlimited Release  
Printed April 1998

**A User's Guide  
for  
BREAKUP: A Computer Code  
for Parallelizing the Overset Grid Approach**

Daniel W. Barnette  
Parallel Computational Sciences department  
Sandia National Laboratories  
Albuquerque, New Mexico 87185-1111  
Email: [dwbarn@cs.sandia.gov](mailto:dwbarn@cs.sandia.gov)

**Abstract**

In this user's guide, details for running BREAKUP are discussed. BREAKUP allows the widely used overset grid method to be run in a parallel computer environment to achieve faster run times for computational field simulations over complex geometries. The overset grid method permits complex geometries to be divided into separate components. Each component is then gridded independently. The grids are computationally rejoined in a solver *via* interpolation coefficients used for grid-to-grid communications of boundary data. Overset grids have been in widespread use for many years on serial computers, and several well-known Navier-Stokes flow solvers have been extensively developed and validated to support their use. One drawback of serial overset grid methods has been the extensive compute time required to update flow solutions one grid at a time. Parallelizing the overset grid method overcomes this limitation by updating each grid or subgrid simultaneously. BREAKUP prepares overset grids for parallel processing by subdividing each overset grid into statically load-balanced subgrids. Two-dimensional examples with sample solutions, and three-dimensional examples, are presented.

## **Acknowledgements**

The author wishes to acknowledge Dr. Curtis C. Ober of Sandia's Parallel Computational Science Department for his help in modifying a two-dimensional incompressible Navier-Stokes code to run on the Intel Paragon with output from BREAKUP.

# Contents

<b>Introduction</b>	6
<b>The Parallel Overset Grid Approach</b>	
a) Overview	7
b) Overset Grid-to-Grid Communications	7
c) Parallel Overset Grid Construction	7
<b>How BREAKUP Prepares Grids for Parallel Processing</b>	
a) Load balance	8
b) Speed-up	9
c) Sample BREAKUP Output	9
d) Construction of Connectivity Tables	13
<b>Running BREAKUP</b>	14
<b>User Options</b>	
a) Option 1 – PEGSUS-formatted Generic Overset Grids	14
b) Option 2 – PLOT3D-formatted Multiple Grids, No Interpolation Coefficients, Subgrid Overlap Required	15
c) Option 3 – PLOT3D-formatted Multiple Grids, No Interpolation Coefficients, No Subgrid Overlap Required; or Internally Generated Grid	15
<b>Input Files</b>	16
<b>Output Files</b>	17
<b>Examples</b>	17
<b>Future Work and Directions</b>	18
<b>Summary</b>	19
<b>References</b>	20
<b>Appendices</b>	
A: BREAKUP Subroutines and Their Description	38
B: List of Variables used in BREAKUP	51
C: Sample Output from BREAKUP (file BREAKUP.OUT)	53
<b>Figures</b>	
1. Overview flow chart for the parallel overset grid method	21
2. Overview flow chart for BREAKUP when overset grid output ...	22
3. Illustration of increase in grid-to-grid and intra-grid communications...	23
4. BREAKUP's subroutines listed in alphabetical order	24
5. BREAKUP calling tree	25
6. Cube illustrating how BREAKUP generates 74 subgrids	26
7. BREAKUP-generated subgrids for a multi-element airfoil grid...	27
8. BREAKUP subgrids for an F16's forward fuselage and intake duct	29
9. Original six overset grids for the Gulf of Mexico, with serial solution	30
10. Sixteen BREAKUP-generated overset grids for the Gulf of Mexico	34
11. One hundred BREAKUP-generated overset subgrids for the Gulf of Mexico	36

**A User's Guide  
for  
BREAKUP: A Computer Code  
for Parallelizing the Overset Grid Approach**

## **Introduction**

The overset grid approach for computing field simulations around geometrically complex configurations has become widely used since its inception in the early 1970's[1]. The attractiveness of the approach lies in the fact that a complex geometry can be divided into its inherently simpler parts while maintaining geometric complexity. Each part is then gridded separately. This is presumed to be much less complicated a process than suitably gridding the entire geometry alone. The grids are computationally re-combined in a solver in such a manner as to give a smooth solution within and across each grid domain. The approach has continued to mature in the serial computing environment. The wide acceptance of the method is reflected in the fact that overset grids have been implemented in well-known compressible as well as incompressible Navier-Stokes flow solvers.

Until the paradigm of parallel computing was developed, one of the major drawbacks to the overset grid approach was the necessity of updating the solutions on each grid separately. This involves sequentially reading each grid into core, updating the flow field one time step, and re-writing the updated solution back to disk storage. The next grid is then read into core and the process repeated. Each grid receives its updated boundary conditions from other grids *via* interpolation coefficients. The coefficients are computed using other codes after the overset grids are generated but before the grids are submitted to a flow solver. Since iterative solvers typically require on the order of thousands of time steps for convergence, wall-clock time can be excessive when grids are swapping in and out of core. Additionally, time spent in queues during run time can result in significant delays for solutions that would require orders of magnitude less wall clock time had the user sole use of the computer.

The drawbacks noted above may be overcome by a parallel approach to the overset grid method. The 'parallelized overset grid approach' discussed herein involves dividing each of the several main grids into an arbitrary number of load-balanced subgrids, with each subgrid assigned to a processor in a distributed or massively parallel computing environment. The previously generated interpolation coefficients for grid-to-grid communications are analyzed and reassigned (not regenerated!) to their corresponding subgrid processor so that each knows which processor will need its updated dependent variables and which processors will be sending updated dependent variables to it. The solvers must be modified to run on parallel computers so that each subgrid can now be updated simultaneously rather than sequentially. This approach leads to significantly faster run times for more complex problems than could have been modeled in the past.

This user's guide discusses the computer code BREAKUP. The BREAKUP code was developed to subdivide overset grids in a manner that load balances the application on parallel computers. The code provides the capability to generate parallel solutions using overset grids in solvers modified for parallel computers. The code has been written so that a minimum of changes will be needed to modify a solver for parallel applications.

## **The Parallel Overset Grid Approach**

### **a) Overview**

An overview flow chart for the parallel overset grid method is presented in Fig. 1. The flow chart illustrates the point that the method was designed to minimally impact the serial overset grid approach. The serial-to-parallel modifications needed to implement the approach involve the requirement that the grids need additional preparation for the parallel solver, and that the solver must be modified *via* message passing calls for use on a parallel computer. The method currently implemented is such that all codes, *i.e.* BREAKUP, visualization codes, and pre- and post-processing codes, run on a workstation. The solver, of course, is run on a parallel computer.

### **b) Overset Grid-to-Grid Communications**

Overset grids require the capability to communicate with overlapped or embedded grids. Currently, PEGSUS 4.0[2] is used to generate interpolation coefficients for grid-to-grid communications where separate grids overlap. Solvers that are PEGSUS-based use the interpolation coefficients and data from other influencing grids to update boundary conditions in overlapped regions on each grid as the governing equations are solved. Users should consult Ref. 2 for more detailed information.

Other codes exist to generate the interpolation coefficients. However, as currently written, BREAKUP is configured to read the output of PEGSUS only.

### **c) Parallel Overset Grid Construction**

BREAKUP may be considered as a preprocessor to PEGSUS-based parallel solvers and, as such, represents the cornerstone to the parallel overset grid approach. An overview flow chart of BREAKUP is presented in Fig. 2. The primary purpose of BREAKUP is to prepare the output of PEGSUS for parallel computing. PEGSUS output includes the overset (known as 'composite' in PEGSUS jargon) grids, the 'iblack' array that flags grid points for special handling in the solver, and an interpolation coefficient file used for grid-to-grid communication. BREAKUP divides each overset grid into subgrids corresponding to the user-specified number of processors on a parallel computer or in a distributed computing environment. The code computes the total number of grid points, computes the average number of grid points per processor, and divides each grid into subgrids such that a near-uniform distribution of grid points exists on each processor. Separate overset grids are divided such that the grid's perimeter-to-area ratio in 2-D, or surface-to-volume ratio in 3-D, is minimized. The subgrids are assigned to individual

processors in a sequential manner with no regard to optimal communication paths between adjacent processors. This means that there is no guarantee that two subgrids from different overset grids requiring high communications costs will lie adjacent or even close to each other to minimize latency. This is an area for further algorithmic development.

Dividing the overset grids into subgrids results in a significant amount of message passing between processors, as illustrated in Fig. 3. This is due to the necessity for grid-to-grid communications as well as intra-grid communication. Intra-grid communication is defined as message passing between subgrids of a unique overset grid. Grid-to-grid communication is defined as message passing between subgrids of separate overset grids. For grid-to-grid communications, the additional boundaries resulting from forming subgrids requires appropriate interpolation coefficients corresponding to each new subgrid. The interpolation coefficients are the same as those computed by the PEGSUS code. No new data are generated in BREAKUP. BREAKUP searches these coefficients and correlates them to the appropriate subgrids.

As mentioned above, intra-grid communication results from overlapping subgrids of an individual, unique overset grid. Adjacent subgrids align point-to-point by definition and, therefore, require no interpolation coefficients *per se*. However, the solver must know where to send and receive boundary information for intra-grid, as well as grid-to-grid, communications. BREAKUP constructs connectivity tables for both grid-to-grid and intra-grid communications such that each processor has the correct data to update its subgrid. The table for grid-to-grid communications contains the processor location of each pair of interpolation coefficients and those points on other processors (*i.e.*, subgrids) which require the coefficients. The solver must be modified to use the tables to update grid boundary data. More detail regarding connectivity tables is given below.

## **How BREAKUP Prepares Grids for Parallel Processing**

Two primary issues in parallel processing are load balance and speed-up. These issues, and how they are dealt with in BREAKUP, will now be considered. Also discussed is the construction of connectivity tables used in the user-modified solver to define processor-to-processor communications.

### **a) Load balance**

Load balancing on parallel processors means that each processor should have the same, or nearly the same, work. In practice, this can be difficult to precisely achieve. The inherent power of the overset grid method lies in the fact that the size of individual overset grids can significantly vary in any given problem.

The approach to load-balancing subgrids in BREAKUP is as follows. First, BREAKUP determines the total number of grid points in all grids and divides by the number of user-specified subgrids (or processors) required. This gives the average number of grid points

per processor. Next, the number of grid points in each individual overset grid is divided by this number to give the total number of processors that will be dedicated to that grid. Of course, this number seldom, if ever, is a whole number. BREAKUP is programmed to sequentially enlarge subgrid boundaries within the said grid to encompass more grid points that remain after calculating the number of processors per grid. This process ensures that the subgrid boundaries remain flexible enough to encompass all grid points. Also, it keeps the impact on load balancing to a minimum while keeping fixed the number of processors assigned to that grid.

If it happens that many grid points are left over, it becomes possible to better load balance the original grids with a larger number of processors. In this case, the code outputs a message to the effect that the grids cannot be optimally subdivided on the specified number of processors. BREAKUP then calculates the nearest number of processors over which the grids would be better load balanced. The user is then queried to either accept the new number and continue or input a new value for the number of processors required.

It is possible that BREAKUP will be given a small enough grid that cannot be subdivided. In this case, BREAKUP will leave the small grid intact. The premise is that, if there are only a few, these relatively small grids will not have a significant impact on the load balancing of the entire set of overset grids. That is, only a few processors with very small grids will have to wait on the many other processors to finish a solution time step.

## **b) Speed-up**

Speed-up occurs by minimizing communications between subgrids and maximizing the computational work performed on each processor. Communication occurs at the grid boundaries and hence is proportional to the grid's surface area. Computational work is performed at each grid point and is therefore proportional to the grid volume. Hence, it is advantageous when subdividing the grid to minimize its surface area and maximize its volume. Once the number of processors assigned to each overset grid is known, BREAKUP determines all the 3-factors of that number and finds the minimum value of the ratio of subgrid surface area to volume.

## **c) Sample BREAKUP output**

An example of how BREAKUP handles load-balancing and speed-up will illustrate the process.

Assume a user has the following six grids, with the grid size indicated by  $j$ ,  $k$ , and  $l$ . Each grid has the total number of points listed.

Grid#	j	k	l	Total Points
1	3	77	53	12243
2	3	249	20	14940
3	3	240	20	14400
4	3	135	20	8100
5	3	234	20	14040
6	3	133	20	7980

The total number of grid points is 71,703. If the user chooses to subdivide these grids into 16 subgrids, the average number of points per processor (rounded to the nearest integer) is 4,481. Dividing the number of grid points for each grid by the average number of gridpoints desired on each processor yields the following information:

Grid #	Total points in grid	Avg. no of grid points per processor	No. of processors for each grid
1	12243	4481	3
2	14940	4481	3
3	14400	4481	3
4	8100	4481	2
5	14040	4481	3
6	7980	4481	2

BREAKUP calculates that three processors will be assigned to Grid #1, three to Grid #2, and so on. This implies, of course, that Grid #1 will be divided into three subgrids, Grid #2 into three subgrids, *etc.* Although this simple calculation informs the user how many subgrids will be formed from each original overset grid, it does not indicate how the subgrids will be formed. To do this, BREAKUP examines all of the three-factors that can be determined from the number of processors assigned for each grid. For Grid #1, BREAKUP determines that the original grid can be subdivided in the following ways.

ITAG	J	K	L
1	1	1	3
2	1	3	1
3	3	1	1

The *ITAG* parameter is for reference only. The row-wise product of the three-factors listed under the *J*, *K*, and *L* columns is always equal to the number of processors assigned to Grid #1. Here, *J*, *K*, and *L* are assumed to be oriented in the grid's *j*, *k*, and *l* direction, respectively, and represent the maximum indices that will be assigned to the subgrids. For example, Grid #1 can be subdivided into 1 subgrid in the grid's own *j* direction, 1 subgrid in its *k* direction, and 3 subgrids in its *l* direction. Any permutation of this can also be made. However, the permutation desired is the one that will yield a minimum surface area for each subgrid and a maximum grid volume. Therefore, each combination

of three-factors, associated with increasing values of *ITAG*, must be examined to give the minimum ratio of subgrid area over subgrid volume.

The surface area of any one subgrid for Grid #1 may be written as

$$surf = 2 \left[ \left( \frac{j \max}{J} \times \frac{k \max}{K} \right) + \left( \frac{k \max}{K} \times \frac{l \max}{L} \right) + \left( \frac{j \max}{J} \times \frac{l \max}{L} \right) \right] \quad (1)$$

where *jmax*, *kmax*, and *lmax* are the dimensions of the original overset grid, and *J*, *K*, and *L* are the number of subgrids into which the grid will be subdivided in their respective directions. Hence, the quantity *jmax/J* will denote the dimension, or number of grid points, of one subgrid in the *j* direction. Note that *J*, *K*, and *L* can take on any permutation of the three-factors listed above. The volume of any of the subgrids is given by

$$vol = \frac{j \max}{J} \times \frac{k \max}{K} \times \frac{l \max}{L}. \quad (2)$$

The area-to-volume ratio is given by

$$\frac{surf}{vol} = 2 \left( \frac{J}{j \max} + \frac{K}{k \max} + \frac{L}{l \max} \right) \quad (3)$$

This ratio is to be minimized to achieve the most efficient configuration for subdividing the original grid. Minimization occurs by substituting the various three-factors for *J*, *K*, and *L* associated with the variable *ITAG*. The ratio is calculated for each permutation. The combination that gives the minimum ratio is selected as the way to subdivide the grid. For Grid #1, BREAKUP calculates the following ratios for each *ITAG* three-factors.

```

Finding min(surf/vol) for:
  Jmax =          3
  Kmax =         77
  Lmax =         53

  itag   surf/vol   min(surf/vol)
    1     0.805848   0.805848
    2     0.782325   0.782325
    3     2.063710   0.782325

```

```

Final values for min(surf/vol) for this zone are:
  itag =          2
  min(surf/vol) = 0.782325

```

Once the particular three-factor is determined that yields the minimum surface-to-volume ratio, then the grid points in the directions corresponding to the appropriate *ITAG* value chosen above are divided by the appropriate *J*, *K*, or *L* value to give the approximate

number of points per subgrid in each direction after selecting the *ITAG=2* permutation. BREAKUP's output for this is as follows.

```
N/Jmax = 1 / 3 ==> approx. 3 pts/subgrid in J direction
M/Kmax = 3 / 77 ==> approx. 26 pts/subgrid in K direction
P/Lmax = 1 / 53 ==> approx. 53 pts/subgrid in L direction
```

where *N*, *M*, and *P* are used to denote the permuted *J*, *K*, and *L* values for *ITAG=2*. Note that 26 points per subgrid in the *K* direction for three subgrids will leave one point too many in that direction. BREAKUP handles this as indicated by the following output.

```
Warning: Kmaxx/M not evenly divisible!
         K-remainder = 1
```

```
BREAKUP will make adjustments in the size of the
subgrids to compensate for the above remainders.
Adjustments will start at the boundaries and
progress inward. Hence, when remainders appear,
outer subgrids will have their dimensions
increased in the corresponding direction.
```

Final subgrid dimensions are:

Grid #	Jcube	Kcube	Lcube	Jmax	Kmax	Lmax	Total
1	1	1	1	3	27	53	4293
2	1	2	1	3	26	53	4134
3	1	3	1	3	26	53	4134

The values for *Jcube*, *Kcube*, and *Lcube* denote the indices given to each subgrid. The values listed for *Jmax*, *Kmax*, and *Lmax* denote the number of points in the corresponding subgrid along each of its sides. Finally, the total number of grid points for each subgrid in Grid #1 is listed in the *Total* column. The totals include grid points on common and overlapped faces between subgrids. It can be seen the values are within approximately 8% of the 'average no. of grid points per processor' value of 4,481 listed above.

To ensure accuracy, BREAKUP calculates the total number of grid points in all of the subgrids, subtracts the grid points in the common or overlapped regions that were formed as indicated above, and compares the final number with the total number of grid points originally computed for the overset grid. The user is notified if any discrepancies exist. In the present example, the number computed from subgrids and the original number match, as they should, and as indicated in the following output.

```
Compare total number of grid points with original
in this zone:
Subtracted common faces in K-direction.
```

```
Original no. of points in this zone = 12243
Calculated no. of points in this zone = 12243
```

The above process is repeated for each overset grid. If the user has input PEGSUS-generated interpolation coefficients, BREAKUP will correlate the original coefficients with the new subgrids for constructing the connectivity tables, discussed below.

Note that the best speed-up in the above sense would be to have the maximum number of grid points in the maximum amount of memory allowed on processors having identical amounts of memory. This may be a realistic expectation for homogeneous compute clusters or massively parallel machines. It is not realistic for heterogeneous compute clusters. Due to possibly widely varying cpu's, cache, memory, bandwidth, *etc.*, heterogeneous clusters present a significant challenge to any code attempting to achieve a high degree of parallelism. BREAKUP currently does not account for variance of any hardware limitations when subdividing grids.

#### d) Construction of Connectivity Tables

Connectivity tables are needed by the solver to determine which processors need to send information to which receiving processors and *vice versa*. These tables contain information to be used for passing information from processor to processor. Shown below are partial sample outputs for connectivity tables as generated by BREAKUP. Two tables are constructed. One is generated for intra-grid communications; the other, for grid-to-grid communications. The table for intra-grid communications will be discussed first.

A sample connectivity table for intra-grid communications is listed below. The first column indicates the appropriate subgrid under attention. The remaining part of the row lists the subgrid range of indices from which messages will be sent. The next row indicates the subgrid and subgrid indices that will receive information from the processor indicated in the line immediately above it. Hence, each pair of rows indicate a subgrid and its boundaries from which data will be sent, and which subgrid and boundary will receive the data. For example, subgrid 1 will communicate with subgrid 2, as indicated in the first two rows of numbers. The next two rows indicate subgrid 2 will communicate with subgrid 3. Next, it is indicated that subgrid 2 will also communicate with subgrid 1, 3 with 2, 4 with 5, and so on.

c	Zone#	J_beg	J_end	J_inc	K_beg	K_end	K_inc	L_beg	L_end	L_inc
	1	1	3	1	27	27	1	1	53	1
	2	1	3	1	2	2	1	1	53	1
	2	1	3	1	28	28	1	1	53	1
	3	1	3	1	2	2	1	1	53	1
	2	1	3	1	1	1	1	1	53	1
	1	1	3	1	26	26	1	1	53	1
	3	1	3	1	1	1	1	1	53	1
	2	1	3	1	27	27	1	1	53	1
	4	1	3	1	85	85	1	1	20	1
	5	1	3	1	2	2	1	1	20	1

A sample connectivity table for grid-to-grid communications is presented below. The first line indicates the number of points (477) in subgrid #1 whose data need to be sent to

other subgrids. The subsequent columns list the three indices (1,22,43) of the grid point in the donor subgrid #1, the three interpolation coefficients needed to compute the data needed, the processor number (#4) to which data will be sent, and the three indices (1,76,20) of the point in the recipient subgrid at which the dependent variables will be interpolated. The first few lines for donor subgrid #2 are shown for continuity. The solver uses these tables to send and receive messages for updating grid boundaries.

```

477      Base subzone #      1
        1  22  43  0.0000000E+00  6.2530622E-02  3.8906133E-01  4  1  76  20
        1  22  43  0.0000000E+00  2.5979275E-01  4.2713684E-01  4  1  73  20
        1  21  43  1.0000000E+00  8.2356793E-01  2.7348068E-01  4  2  79  20
        2  21  43  1.0000000E+00  9.1715431E-01  3.2546192E-01  4  3  78  20
          .
          .
          .
1035    Base subzone #      2
        1  15  46  0.0000000E+00  2.9964754E-01  9.7948408E-01  4  1  12  20
        1   6  35  1.0000000E+00  2.9722536E-01  6.7863387E-01  4  2  28  20
        1  15  48  0.0000000E+00  5.0462323E-01  9.6113777E-01  4  1   5  20
        2  16  49  1.0000000E+00  5.2282799E-02  3.2107067E-01  4  3   2  20

```

## Running BREAKUP

BREAKUP has been modularly constructed to perform the task of preparing multiple overset grids for parallel processing. This allows the user to more easily run and, if necessary, modify the code. BREAKUP's subroutines are listed in alphabetical order in Fig. 4. A calling tree is presented in Fig. 5 for the reader's reference.

A brief description of each subroutine is listed in Appendix A. The purpose of each subroutine is listed, as well as the calling and called subroutines.

A variable list is presented in Appendix B. A brief description of the more significant variables is given.

A complete BREAKUP output listing for a sample run is presented in Appendix C. Part of the sample output listings noted above were taken from this appendix. The grids used in the appendix correlate to Fig. 10, to be discussed.

When run, BREAKUP presents the user with three options for producing subgrids. The listings shown below duplicate the output from the code upon startup. The options should be self-explanatory. Note that the only significant input required from the user is the number of processors over which the original multiple overset grids must be subdivided.

## User Options

### a) Option 1 – PEGSUS-formatted Generic Overset Grids

This is the most powerful option. Output from the PEGSUS code, which includes the grids, interpolation coefficients, and other necessary data, is read in. The grids are then

subdivided, connectivity tables constructed, and the data output for parallel processing. It also allows leaving the original grids intact but constructing connectivity tables for parallel processing. This implies that each overset grid would reside on one processor with no intra-grid message passing needed.

1) BREAKUP OVERSET GRIDS WITH INTERPOLATION COEFFICIENTS FROM PEGSUS OUTPUT

You have grids for which global PEGSUS output is available for grid-to-grid communication.

If selected, this option has two sub-options:

- a) BREAKUP will subdivide the original grids into subgrids which will communicate using subsets of the original global interpolation coefficients. You will specify the total number of processors over which you want to breakup the grids. This option is typically intended for solving the flow field on the subgrids using a parallel computer.
- b) BREAKUP will leave original grids intact and will not generate subgrids; each grid is written to a separate file for the purposes of each residing on one processor of a parallel computer.

**b) Option 2 – PLOT3D-formatted Multiple Grids, No Interpolation Coefficients, Subgrid Overlap Enforced**

This is a versatile option. It was envisioned that an option was needed so that the user could prepare grids for parallel processing without necessarily having PEGSUS output. Hence, the user could examine the output of BREAKUP, for example, to aid in determining how the code subdivides the grids for a given number of processors.

The original overset grids are read in PLOT3D[3] format instead of the PEGSUS format of Option 1. The subgrids of each separate grid are overlapped one grid cell by default. The overlap extent can be changed in the code internally.

2) BREAKUP PLOT3D MULTI-BLOCK GRIDS; OVERLAP SUBGRIDS

You have one or more grids in a multiblock PLOT3D file that need to be subdivided; no PEGSUS interpolation coefficients are available or needed. BREAKUP will subdivide the original grids into subgrids. The subgrids will be modified to overlap for point-to-point communications.

**c) Option 3 – PLOT3D-formatted Multiple Grids, No Interpolation Coefficients, No Subgrid Overlap Enforced; or Internally Generated Grid**

This option includes Option 2 but without the overlap. It also allows the user to run BREAKUP as a stand-alone code without the need for externally generated grids. Sub-option a) internally generates a cubed grid. The user can then input any number of

processors to divide the cube into the specified number of subgrids. This conveniently allows experimentation with the code 'right out of the box.' Suboption b) requires an externally generated PLOT3D-formatted file.

3) BREAKUP INTERNALLY-GENERATED GRID OR PLOT3D MULTI-BLOCK GRIDS;  
SUBGRIDS ARE NOT OVERLAPPED

You want to see how BREAKUP will create subgrids. BREAKUP will generate a cube grid for you, or you have PLOT3D-format grids to be read in. In either case, you specify how many subgrids you want. Grids read in will not be made to overlap and PEGSUS data (even if available) will not be needed or used.

If selected, this option has two sub-options:

- a) BREAKUP will generate a cube grid (current dimensions 51x51x51); the user can then specify how many subgrids to generate from the original cube grid.
- b) BREAKUP will read a multi-block PLOT3D file; the user specifies how many subgrids in which to subdivide the original grids. These options provide the ability to observe how BREAKUP will work on grid(s) without details associated with Options 1 and 2. These are also useful options for making Vugraphs.

## Input Files

The options discussed above require either PEGSUS files or PLOT3D grid files for input to BREAKUP. Particulars of these formats are listed below.

PEGSUS output (see User Option 1, above) used as input to BREAKUP consists of the following files.

1. INTOUT            -interpolation coefficient file
2. COMPOUT         -composite (overset) grid file
3. IBPLOT           -iblack file

PLOT3D formatted files (see User Options 2 and 3b, above) must be named as follows for BREAKUP to read the files. The local directory is searched for the files in the order given. Hence, if the user has one grid file named XY.FMT (file-read sequence #3, below) and another named G.FMT (file-read sequence #9, below), BREAKUP will read only the former. BREAKUP is programmed to determine the file format (*e.g.*, formatted, binary, multiple grids, *etc.*) automatically.

1. X.FMT
2. X.DAT
3. XY.FMT
4. XY.DAT
5. XYZ.FMT
6. XYZ.DAT

7. GRID.FMT
8. GRID.DAT
9. G.FMT
10. G.DAT

## Output Files

Output files from BREAKUP are listed below with brief explanations. Not all may be output during a run session since the particular options to do so may not be chosen by the user.

- |     |                  |  |
|-----|------------------|--|
| 1.  | UNRAVELED        | main output file to show user what BREAKUP did   |
| 2.  | BREAKUP.OUT      | condensed version of screen output   |
| 3.  | POINTS_OUT       | PLOT3D file for interpolation and boundary points  |
| 4.  | MESH_OUT         | PLOT3D file for the 'iblack'ed mesh  |
| 5.  | 3D_OVERSET_TABLE | file containing base (points used for interpolation) and target zones (points to be interpolated), indices, interpolation coefficients |
| 6.  | 2D_OVERSET_TABLE | decimated version of file 3D_OVERSET_TABLE for two-dimensional problems  |
| 7.  | GRIDS2D          | PLOT3D file for all 2-D grids or subgrids  |
| 8.  | GRIDS3D          | PLOT3D file for all 3-D grids or subgrids  |
| 9.  | 3D_PATCH_TABLE   | table of 3-D grid overlap links between subgrids of an overset grid  |
| 10. | 2D_PATCH_TABLE   | table of 2-D grid overlap links between subgrids of an overset grid  |
| 11. | GRID_ORIGINAL.G  | Original grid file in subroutine   |
| 12. | GRID_DIVIDED.G   | PLOT3D file of subgrids generated by BREAKUP   |
| 13. | 2Dxxxx.FMT       | 2D subgrid files where each subgrid is written to a separate file; xxxx = 0000 to 9999   |
| 14. | 3Dxxxx.FMT       | 3D subgrid files where each subgrid is written to a separate file; xxxx=0000 to 9999   |

## Examples

The first example showing the utility of BREAKUP is presented in Fig. 6. A cube was internally generated and arbitrarily subdivided into 74 subgrids. The plot corresponds to the output of User Option 3a, discussed above.

Another example demonstrating the power of BREAKUP is illustrated in Fig. 7. The five grids generated around a multi-element airfoil are subdivided by BREAKUP into 16 grids for use on the corresponding number of processors. The original grids are presented in Fig. 7a and the divided grids in Fig. 7b. Presented in Fig. 7c is a close-up view of the solution. Smooth streamlines, even in recirculating regions, indicate that message passing between subgrids is behaving as expected. Note from this figure that BREAKUP did not

subdivide the original grid around the slat. BREAKUP determined that the number of grid points for this grid were too few to subdivide further, given the number of user-specified processors. Parallel solutions for the airfoil were generated using a modified version of INS2D[4-6]. Plots were generated using TECPLOT[7].

The next example, shown in Fig. 8, is the for an F-16 forward fuselage and intake duct as obtained from Lockheed-Martin. Presented in Fig. 8a are the original 19 grids. Illustrated in Fig. 8b are the results of subdividing the original 19 grids into 32 subgrids.

The last example, Figs. 9-11, involves two-dimensional ocean current solutions on coastal grids for the Gulf of Mexico and Greater Antilles islands. None of the solutions were run to convergence since this representation is non-physical. The main objective was to determine whether BREAKUP was properly assigning subgrid interpolation coefficients to the processors. The original grids are presented in Fig. 9a. The solution on the original grids is presented in Fig. 9b-d showing velocity vectors, a close-up of the velocity vectors around Cuba, and streamlines, respectively. The original grids were then subdivided into a total of 16 subgrids as presented in Fig. 10a. As expected, BREAKUP subdivided the grids so that approximately the same number of grid points would reside on each processor. A close-up solution for the velocity vectors around Cuba is shown in Fig. 10b. Finally, the original grids were divided into 100 subgrids as shown in Fig. 11a. A close-up velocity vector plot is presented in Fig. 11b. All indications are that, regardless of the number of processors, BREAKUP is generating correct connectivity tables for use in parallel computers. The sample BREAKUP output listed in Appendix C was generated for one of the Gulf simulations noted above.

### **Future Work and Directions**

Any future work on BREAKUP should include coding to aid the user in specifying solver boundary conditions on the subgrids. Specifying boundary conditions by “cut-and paste” editing methods can be a daunting task when preparing overset grids for a multi-thousand processor computer.

Another area of research should be developing the capability within BREAKUP to determine which processors require the highest amount of true interpolated data and to locate these processors near or adjacent to each other.

Modifications should be made to BREAKUP to help implement solvers on heterogeneous, clustered, parallel processors and shared-memory multi-processor workstations. There is a significant amount of work to be done in this area.

An interesting concept would be for BREAKUP to optimize its output based on each processor’s configuration. BREAKUP would need to know or to determine a parallel configuration’s processors, memories, bandwidths, latencies, cache capabilities, *etc.* Obviously, for this process to be automated, BREAKUP would need to run on a machine able to access all processors to be used in the computations. This would not always be possible.

Commensurate with further BREAKUP development should be the task of modifying other overset-grid-based solvers to run on parallel computers and establishing visualization needs for parallel overset-grid solutions.

### **Summary**

The proof of concept in generating high-performance, high-resolution solutions using overset grids in parallel solvers has been demonstrated. Use of overset grids on massively parallel computers will increase users' capabilities to solve challenging problems.

Overset grid technology appears to be well suited for a parallel environment. However, research needs to be conducted for more efficient implementation. This is particularly true due to the ever-changing paradigms and implementations of parallel computing. It is expected that this effort would benefit a wide range of applications important to US industry as well as government laboratories and academia.

## References

1. J. L. Steger, F. C. Dougherty, and J. A. Benek, "A Chimera Grid Scheme," *Advances in Grid Generation*, K. N. Ghia and U. Ghia, eds., ASME FED-Vol. 5, June 1983.
2. N. E. Suhs and R. W. Tramel, "PEGSUS 4.0 User's Manual," AEDC-TR-91-8, Calspan Corporation / AEDC Operations, Arnold AFB, Tennessee, USA.
3. P. Walatka, P. G. Buning, L. Pierce, and P. A. Elson, "PLOT3D User's Manual," NASA Technical Memorandum 101067, March 1990.
4. S. E. Rogers and D. Kwak, "An Upwind Differencing Scheme for the Steady-State Incompressible Navier-Stokes Equations," NASA TM 101051, November 1988. Published in *Journal of Applied Numerical Mathematics*, Vol. 8, 1991, pp. 43-64.
5. S. E. Rogers and D. Kwak, "An Upwind Differencing Scheme for the Time Accurate Incompressible Navier-Stokes Equations," *AIAA Journal*, Vol. 28, No. 2, February, 1990, pp. 253-262.
6. S. E. Rogers, D. Kwak, and C. Kiris, "Numerical Solution of the Incompressible Navier-Stokes Equations for Steady-State and Time-Dependent Problems," *AIAA Journal*, Vol. 29, No. 4, April 1991, pp. 603-610.
7. "TECPLOT – Interactive Data Visualization for Scientists and Engineers, Version 6.0 User's Manual," Amtec Engineering, Inc., P.O. Box 3633, Bellevue, WA.

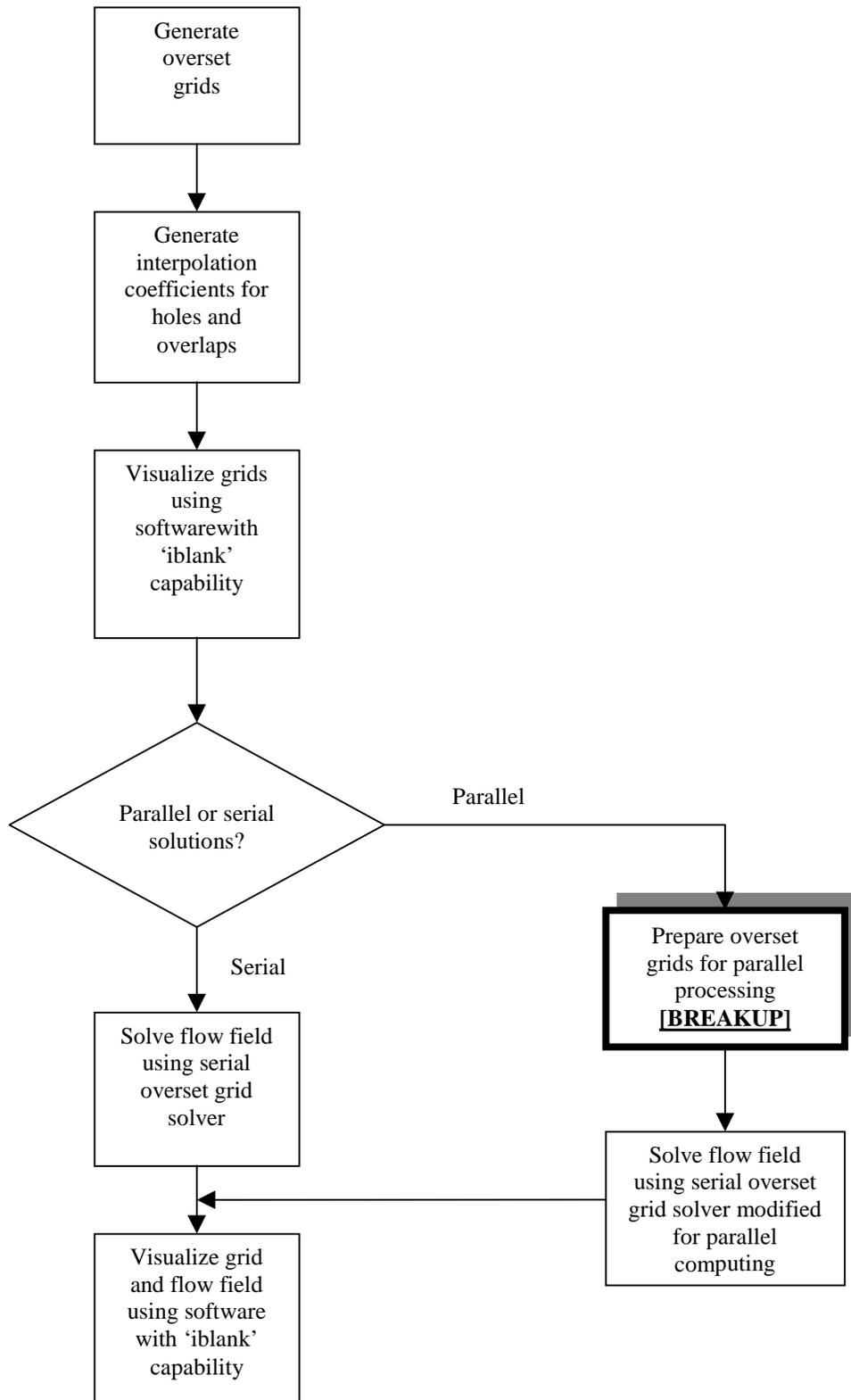


Figure 1. Overview flow chart for the parallel overset grid method.

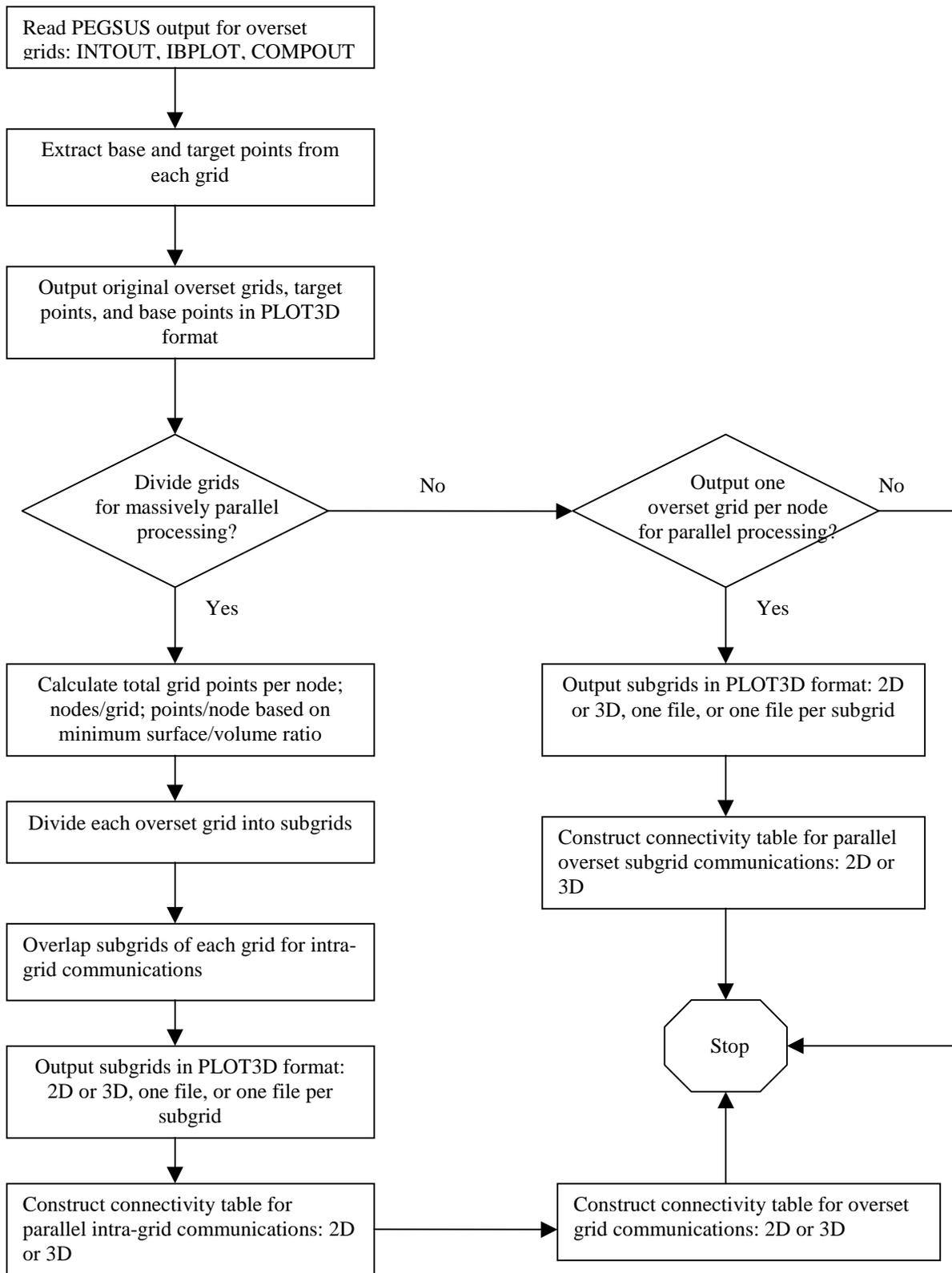
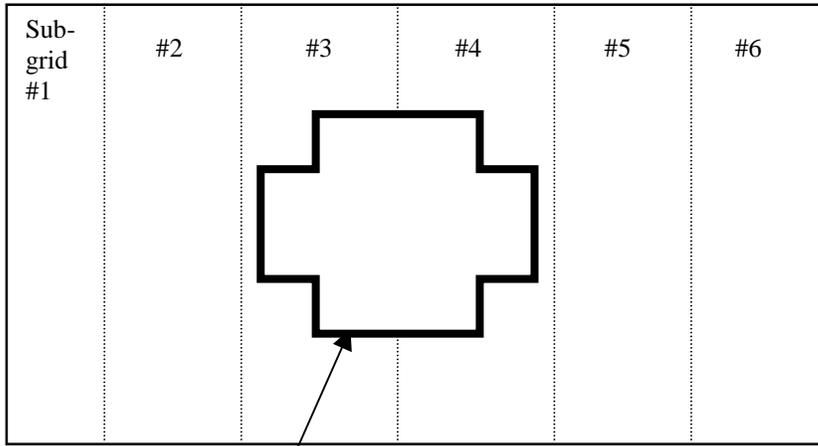
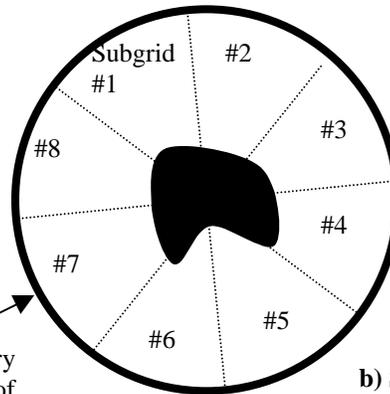


Figure 2. Overview flow chart for BREAKUP when overset grid output from PEGSUS is used as input.

**a) Subdivided Background Grid**

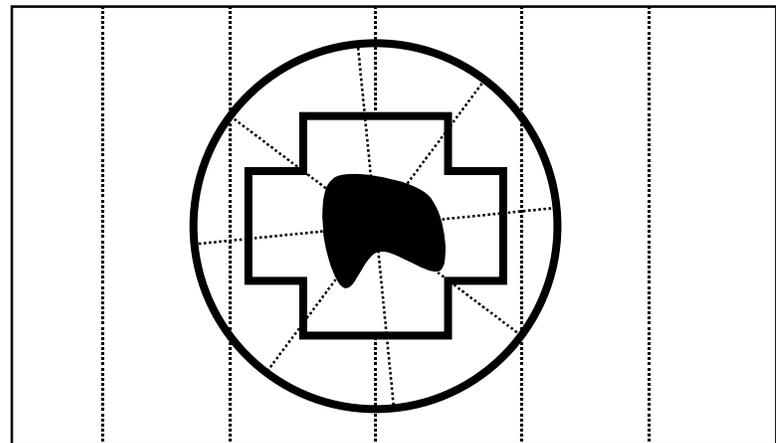
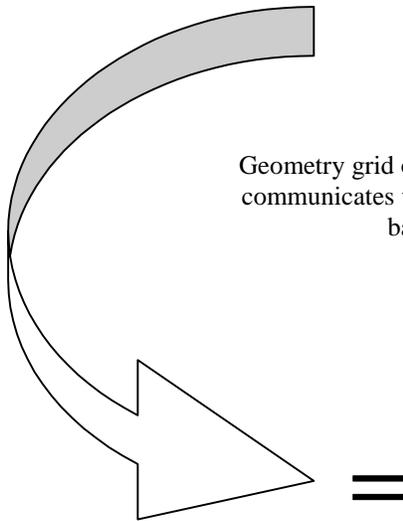


Background grid interior boundary (hole boundary) communicates with interior of geometry grid



**b) Subdivided Grid Around Geometry of Interest**

Geometry grid outer boundary communicates with interior of background grid



**c) Overset grids**

- Intra-grid communication occurs across dashed boundaries within each grid; no interpolation coefficients required due to point-to-point overlap
- Grid-to-grid communication occurs across grid boundaries, represented by heavy lines, *via* interpolation coefficients

Figure 3. Illustration of increase in grid-to-grid and intra-grid communications when applying BREAKUP to overset grids.

```
c Subroutines (alphabetical order)
c
c 1. average
c 2. base_search
c 3. break
c 4. breakup_for_fun
c 5. breakup_grids
c 6. chooser
c 7. decimate
c 8. footer
c 9. get_global_index
c 10. get_zone
c 11. global_indices
c 12. grid_point_comparison
c 13. header
c 14. indexx
c 15. info
c 16. link_overlap
c 17. link_overset
c 18. min_surf_vol_ratio
c 19. nobreak
c 20. outlplanetoplt3d_break
c 21. outlplanetoplt3d_nobreak
c 22. outlplanetoproc_break
c 23. outlplanetoproc_nobreak
c 24. outallplanestoplt3d_break
c 25. outallplanestoplt3d_nobreak
c 26. outallplanestoproc_break
c 27. outallplanestoproc_nobreak
c 28. patch_2d
c 29. read_compout
c 30. read_grid_header
c 31. read_ibplot
c 32. read_intout
c 33. read_zone
c 34. read_zone_header
c 35. sort
c 36. subgrid_dimensions
c 37. target_search
c 38. write_base_target_points
c 39. write_grids
c 40. write_subgrids
c 41. write_unravel
c
```

Figure 4. BREAKUP's subroutines listed in alphabetical order.

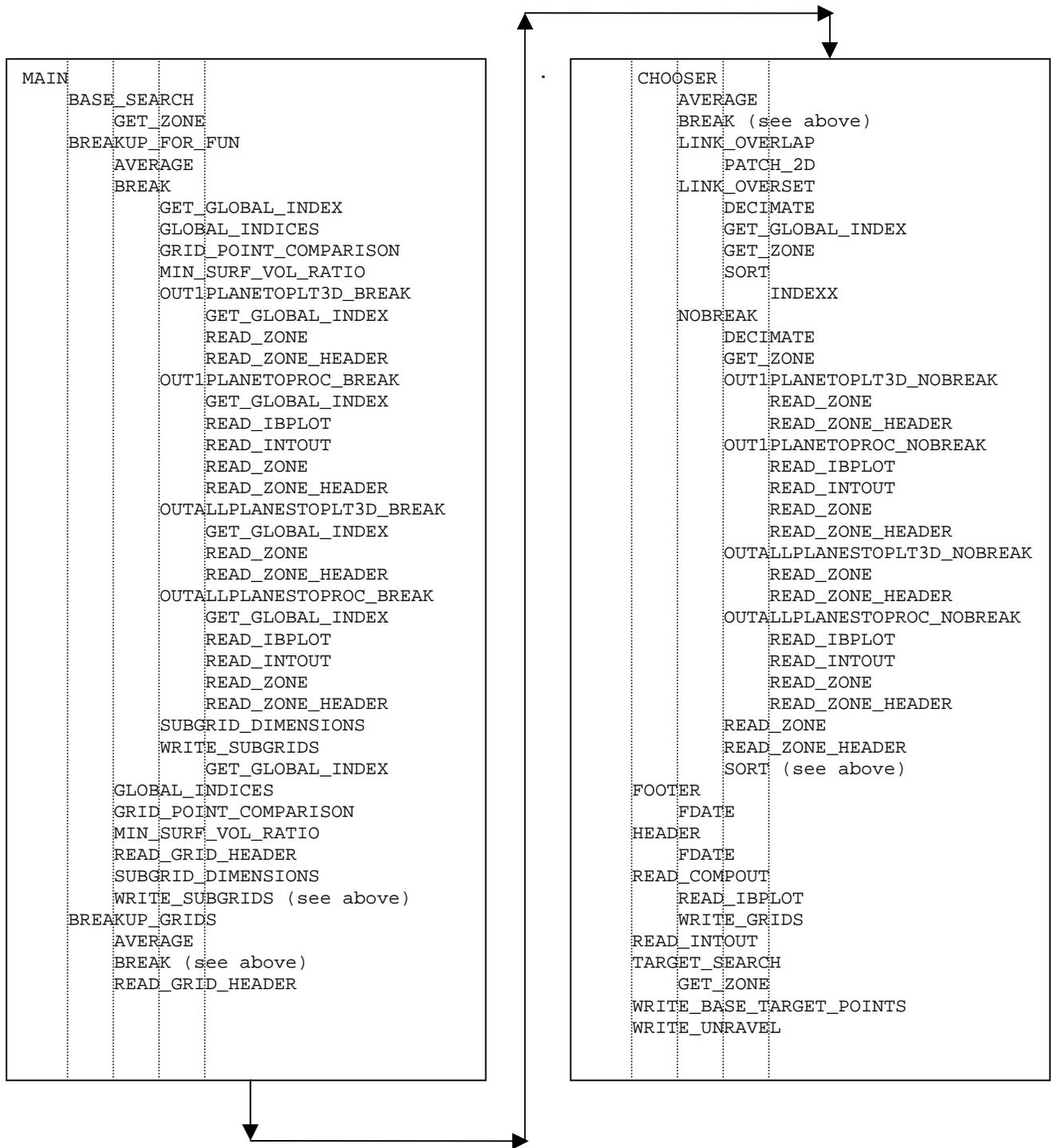


Figure 5. BREAKUP calling tree.

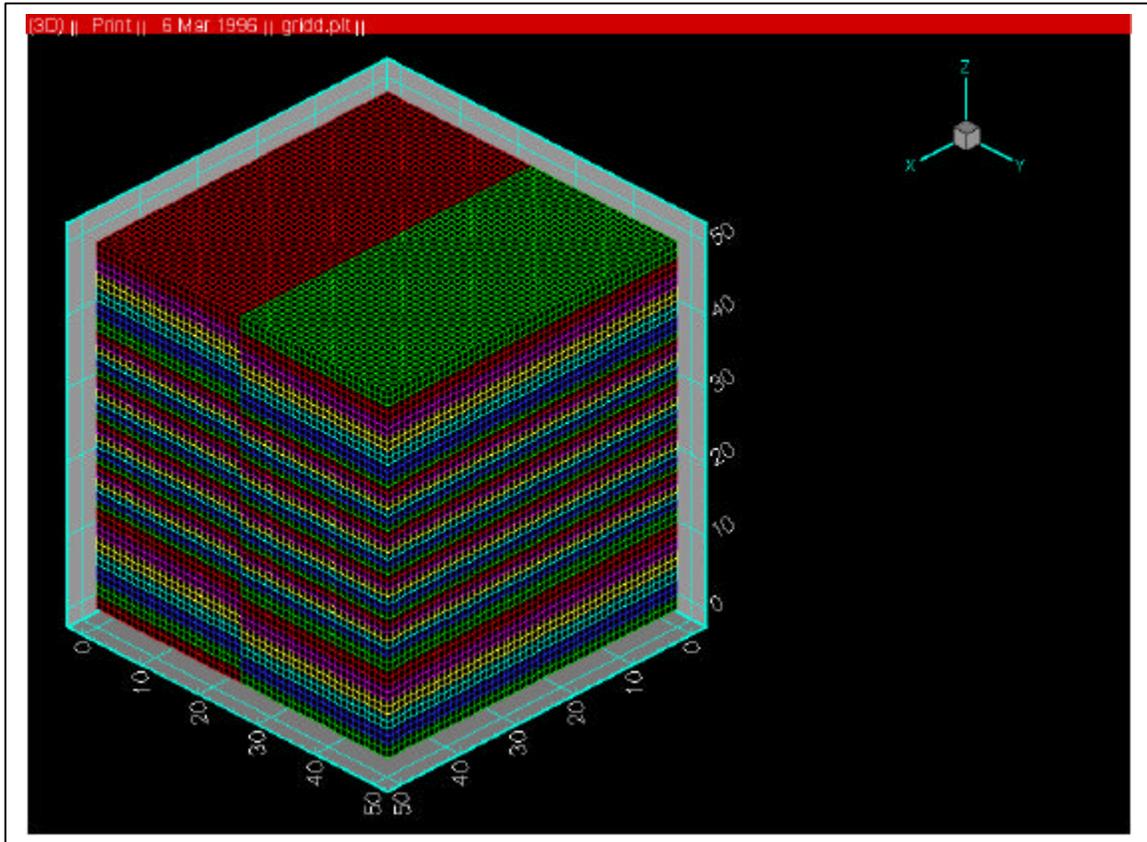
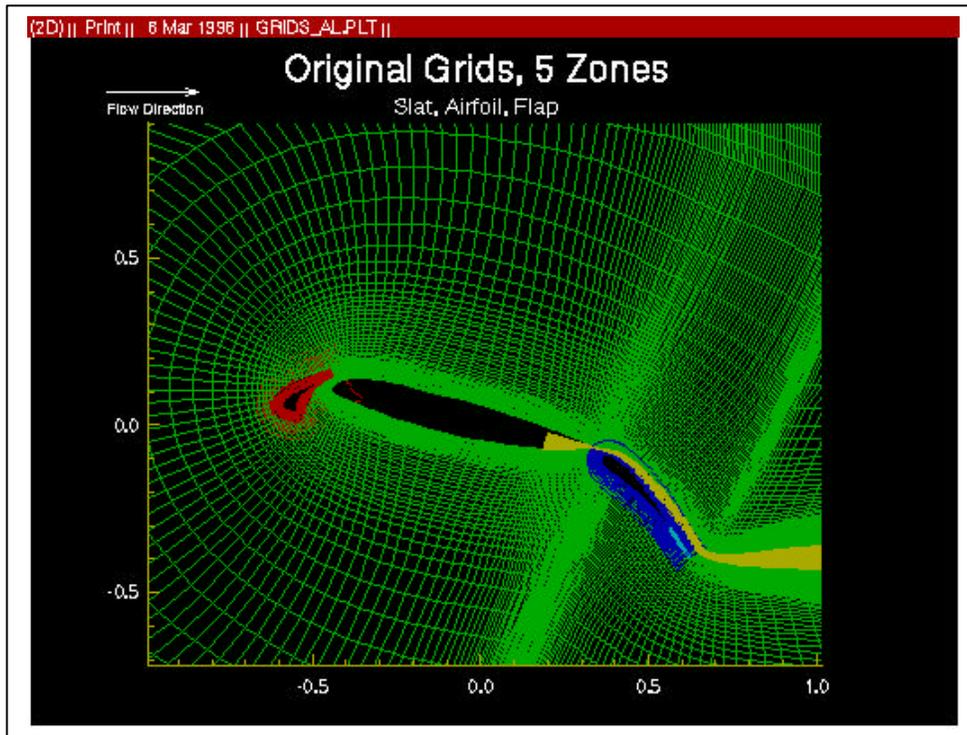
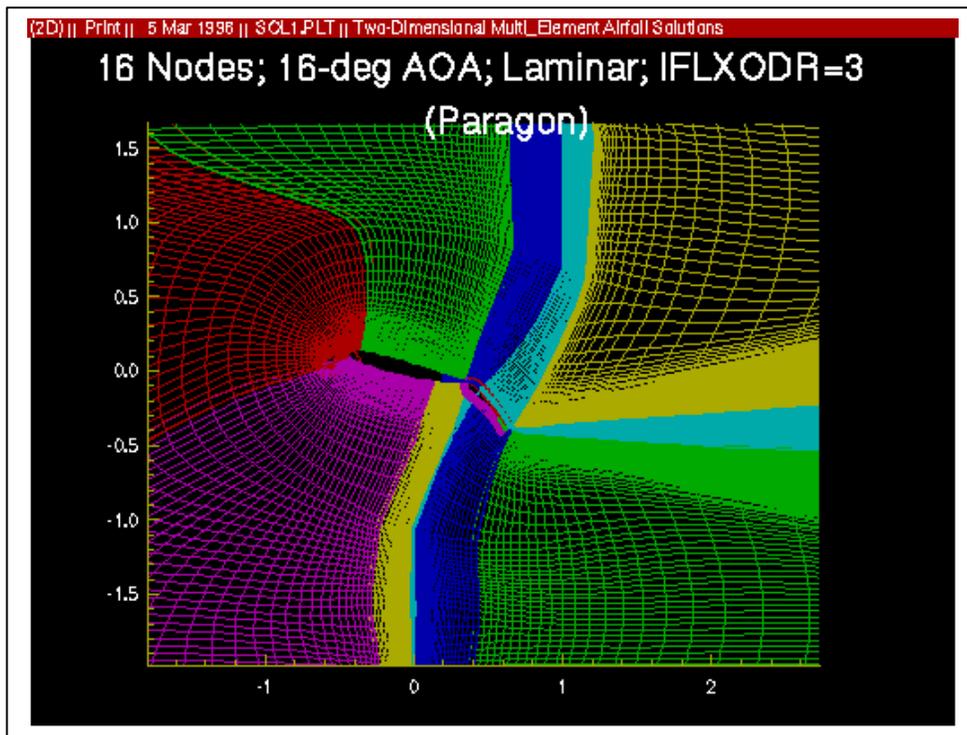


Figure 6. Cube illustrating how BREAKUP generates 74 subgrids.

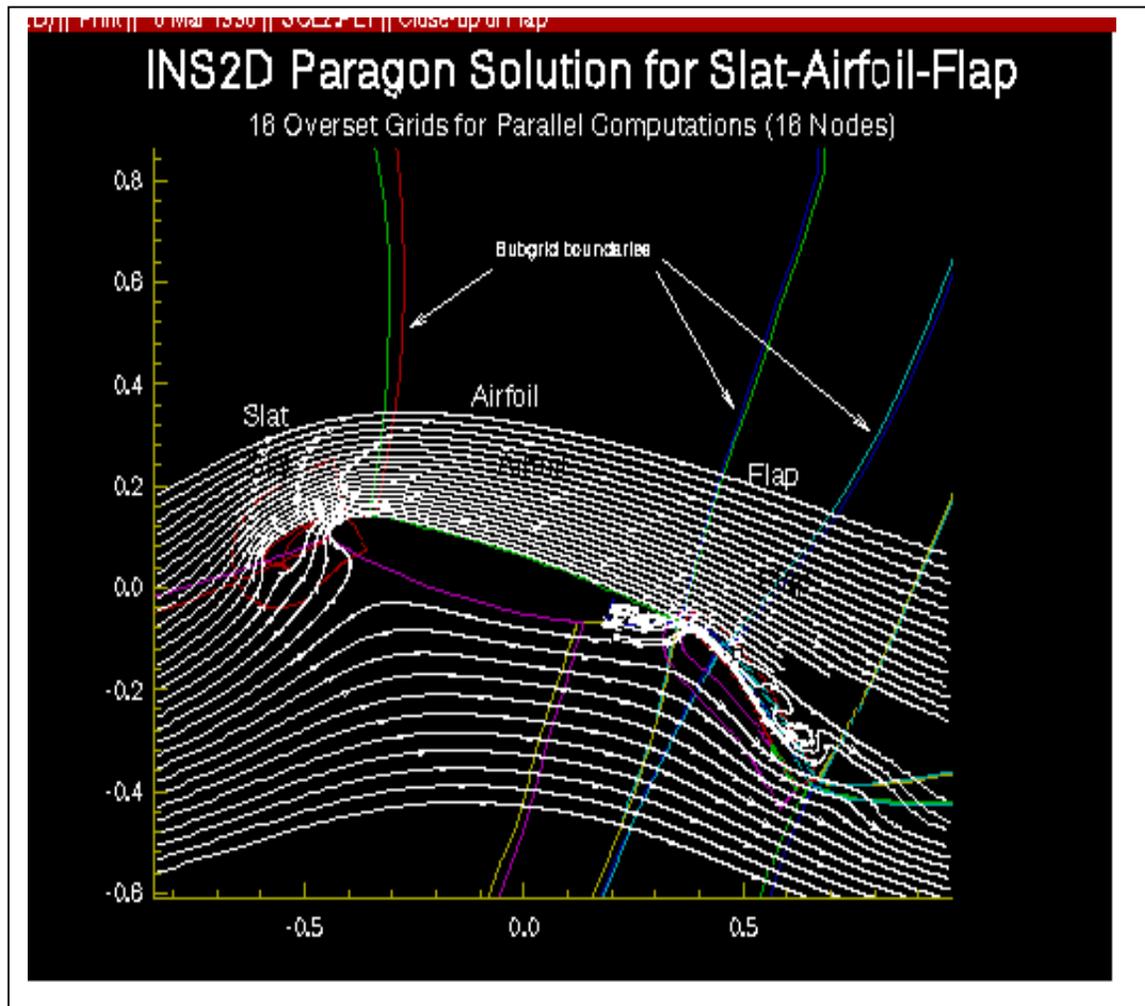


a) Original grids



b) 16 subgrids from BREAKUP

Figure 7. BREAKUP-generated subgrids for a multi-element airfoil grid, with solution.



c) Streamlines for 16-node solution

Figure 7. Concluded.

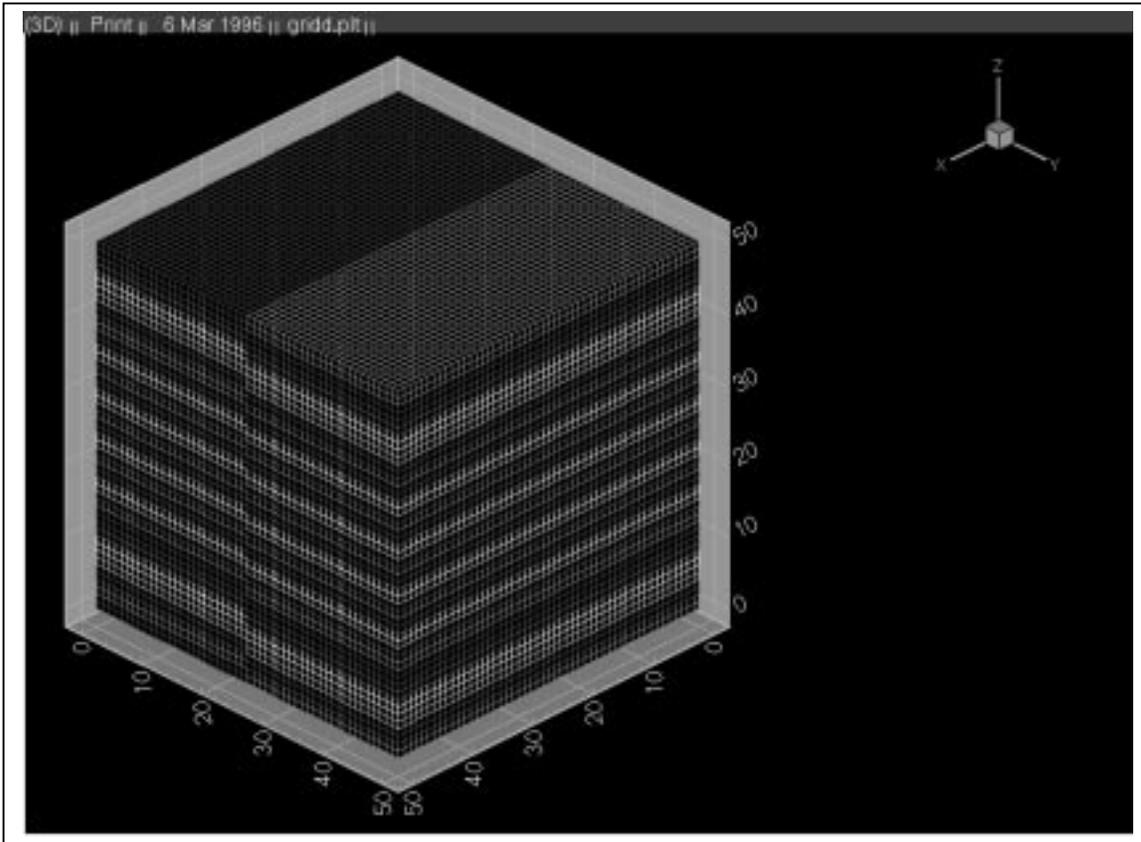
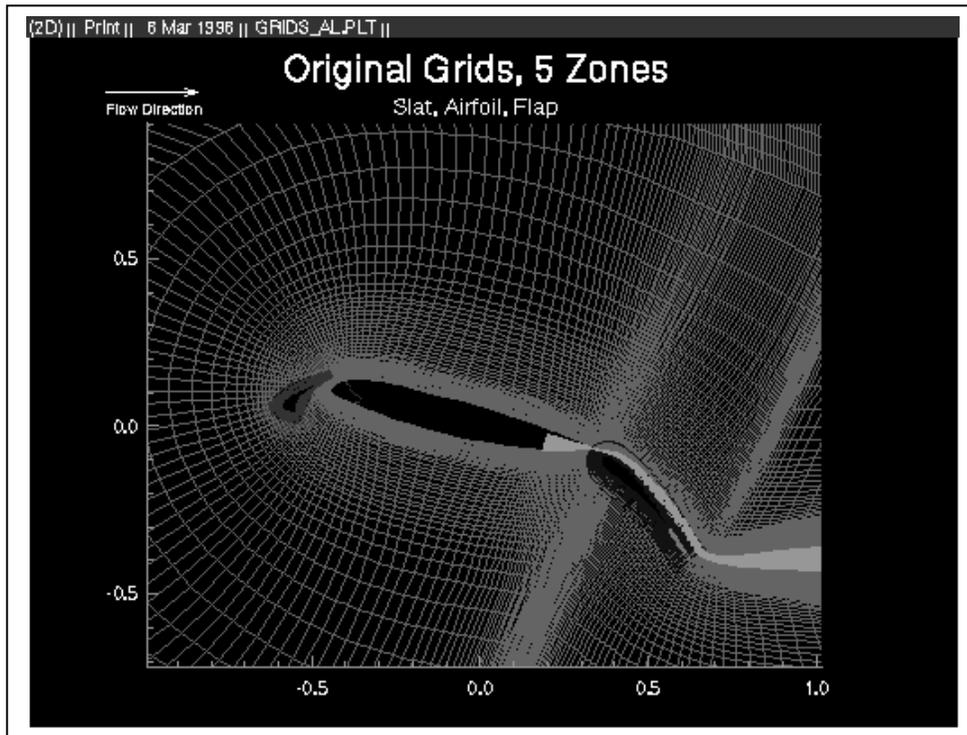
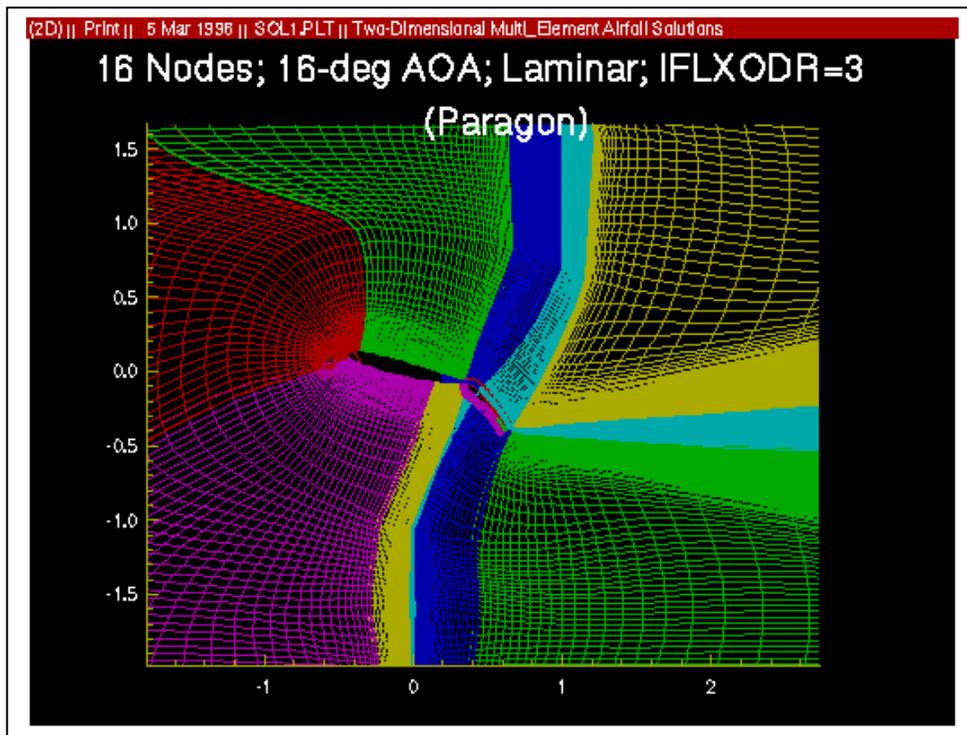


Figure 6. Cube illustrating how BREAKUP generates 74 subgrids.

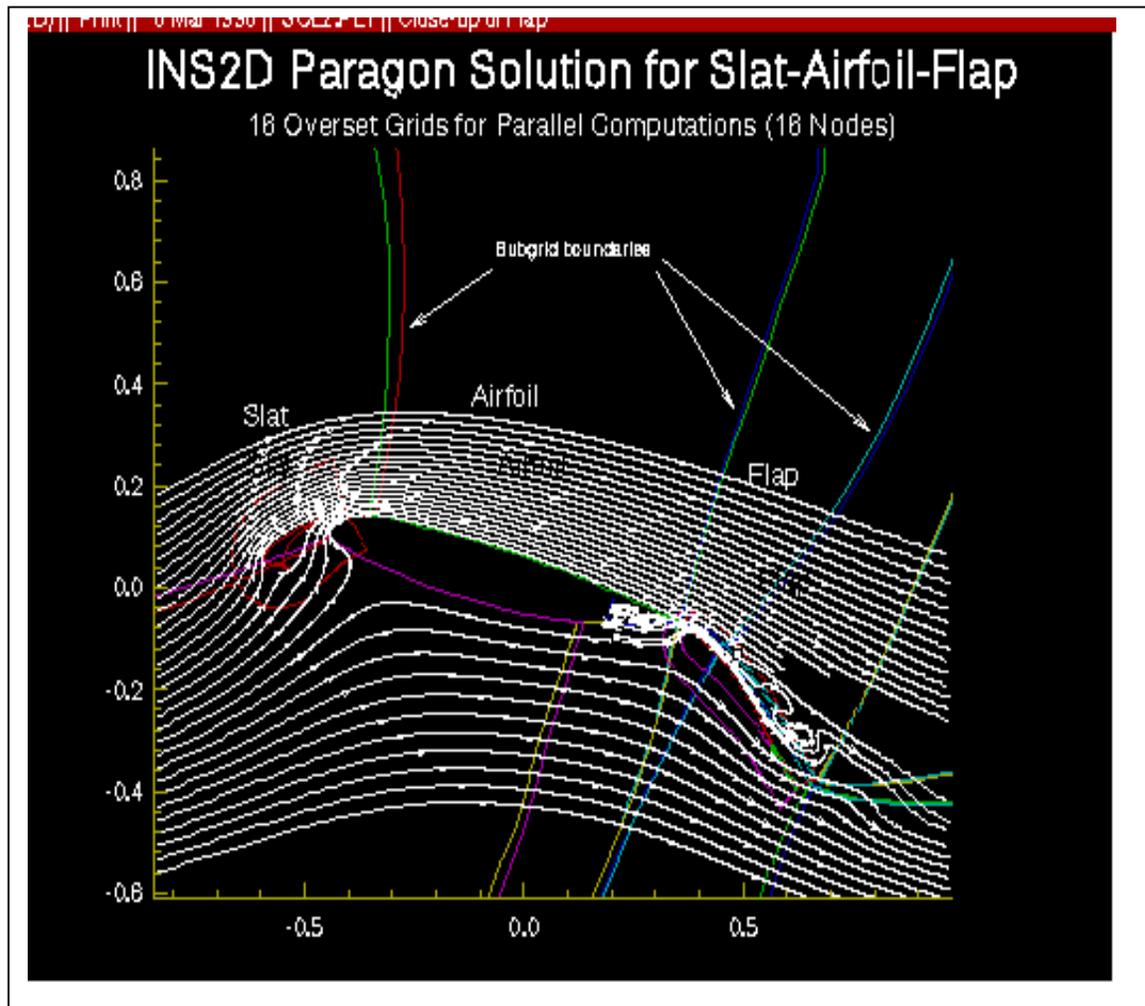


a) Original grids



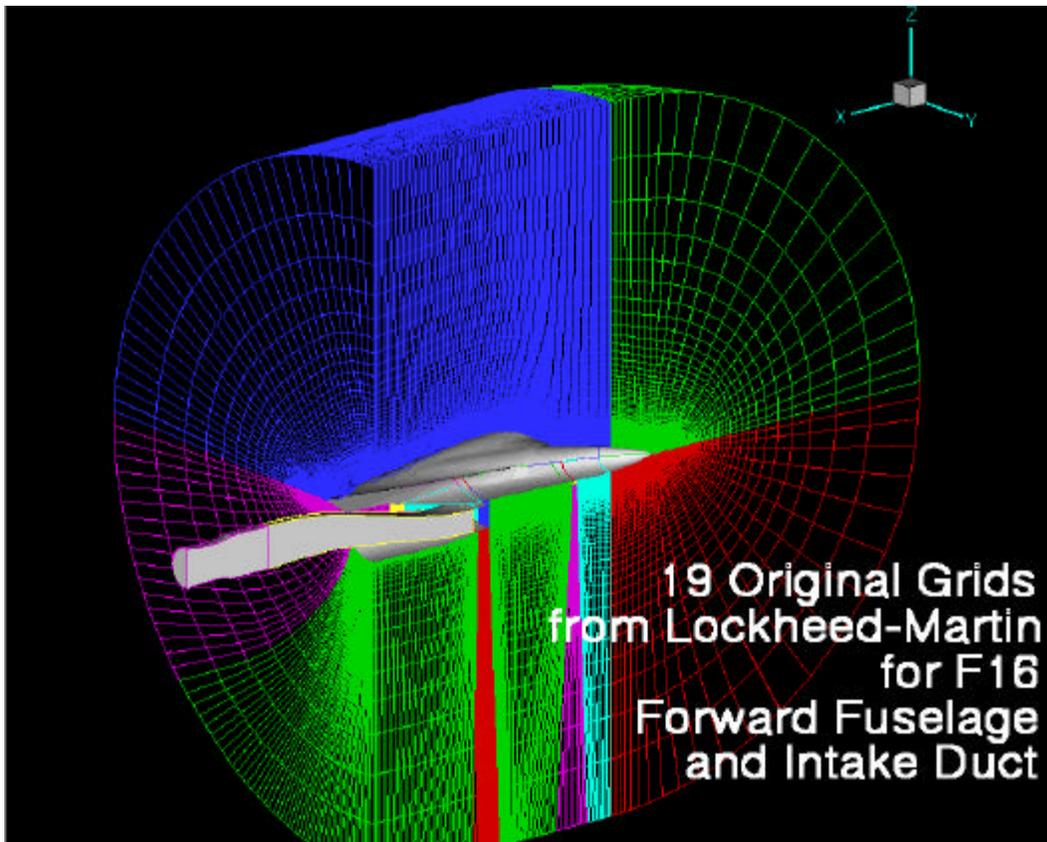
b) 16 subgrids from BREAKUP

Figure 7. BREAKUP-generated subgrids for a multi-element airfoil grid, with solution.

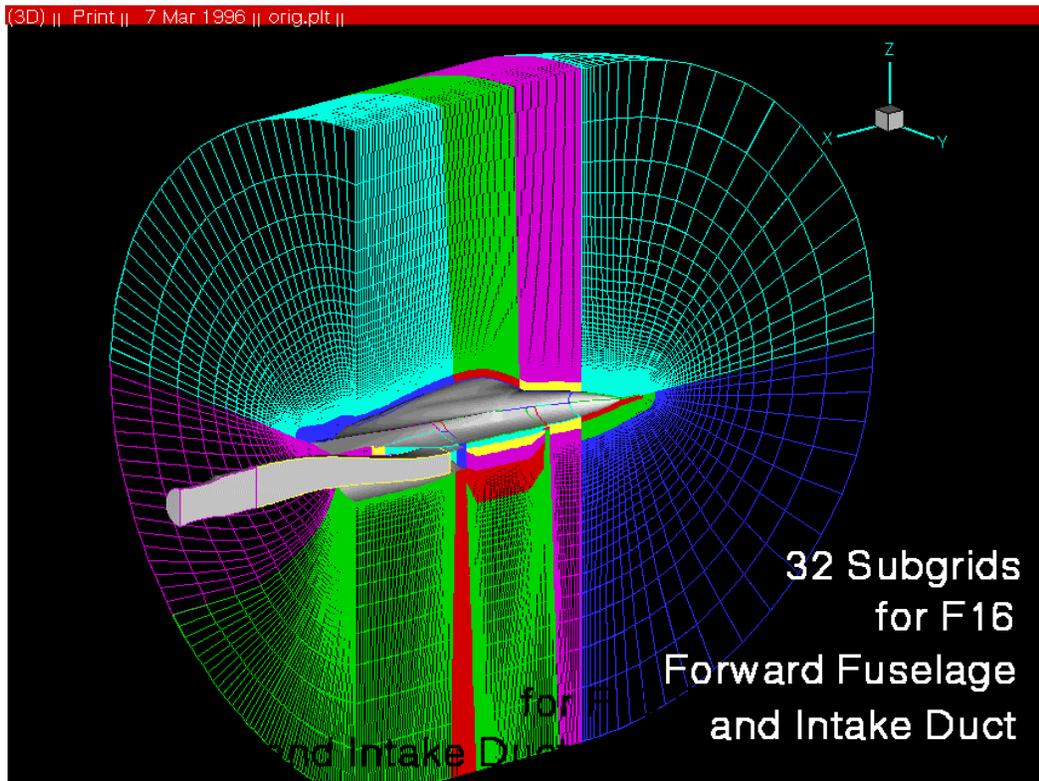


c) Streamlines for 16-node solution

Figure 7. Concluded.

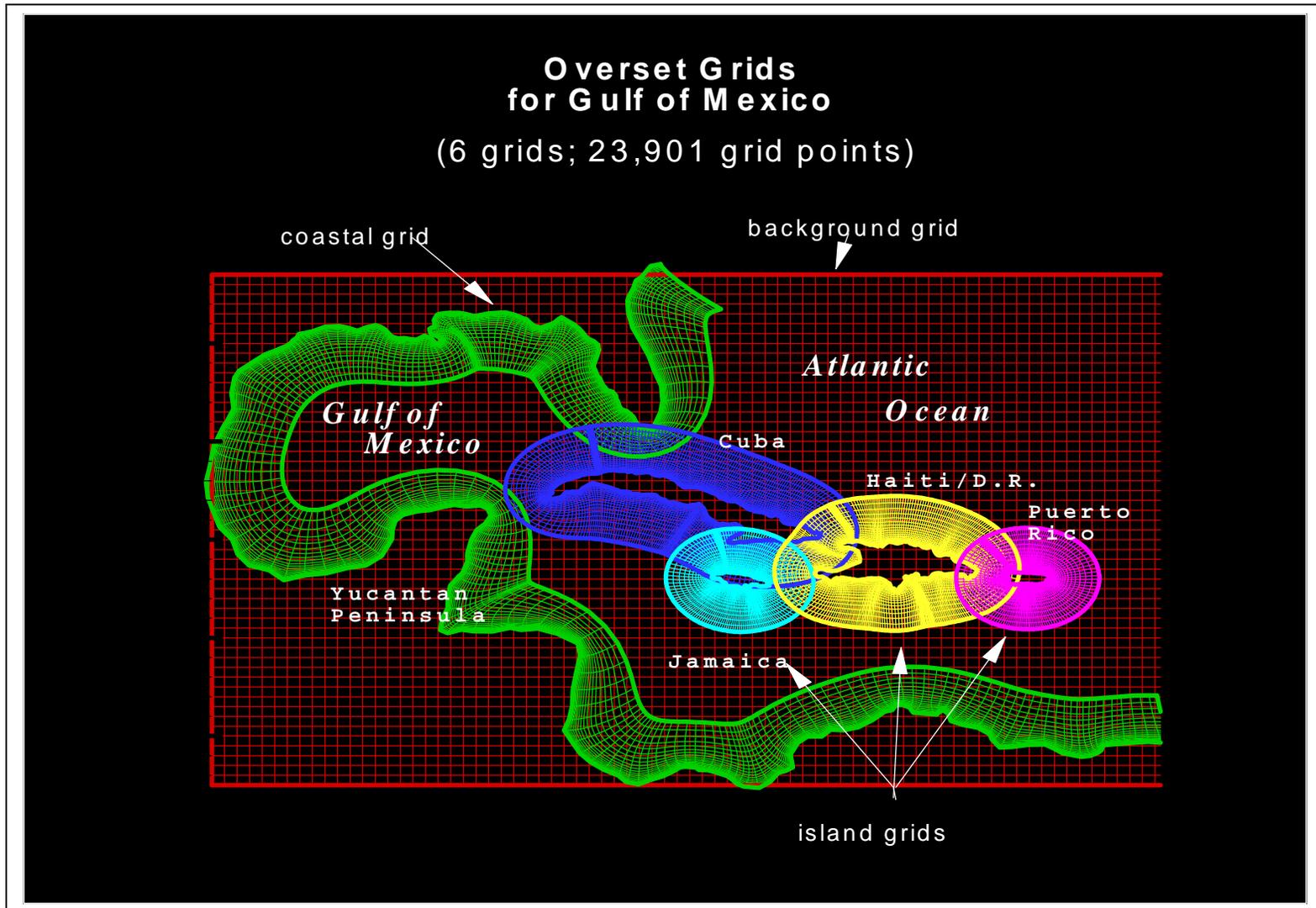


a) 19 original grids



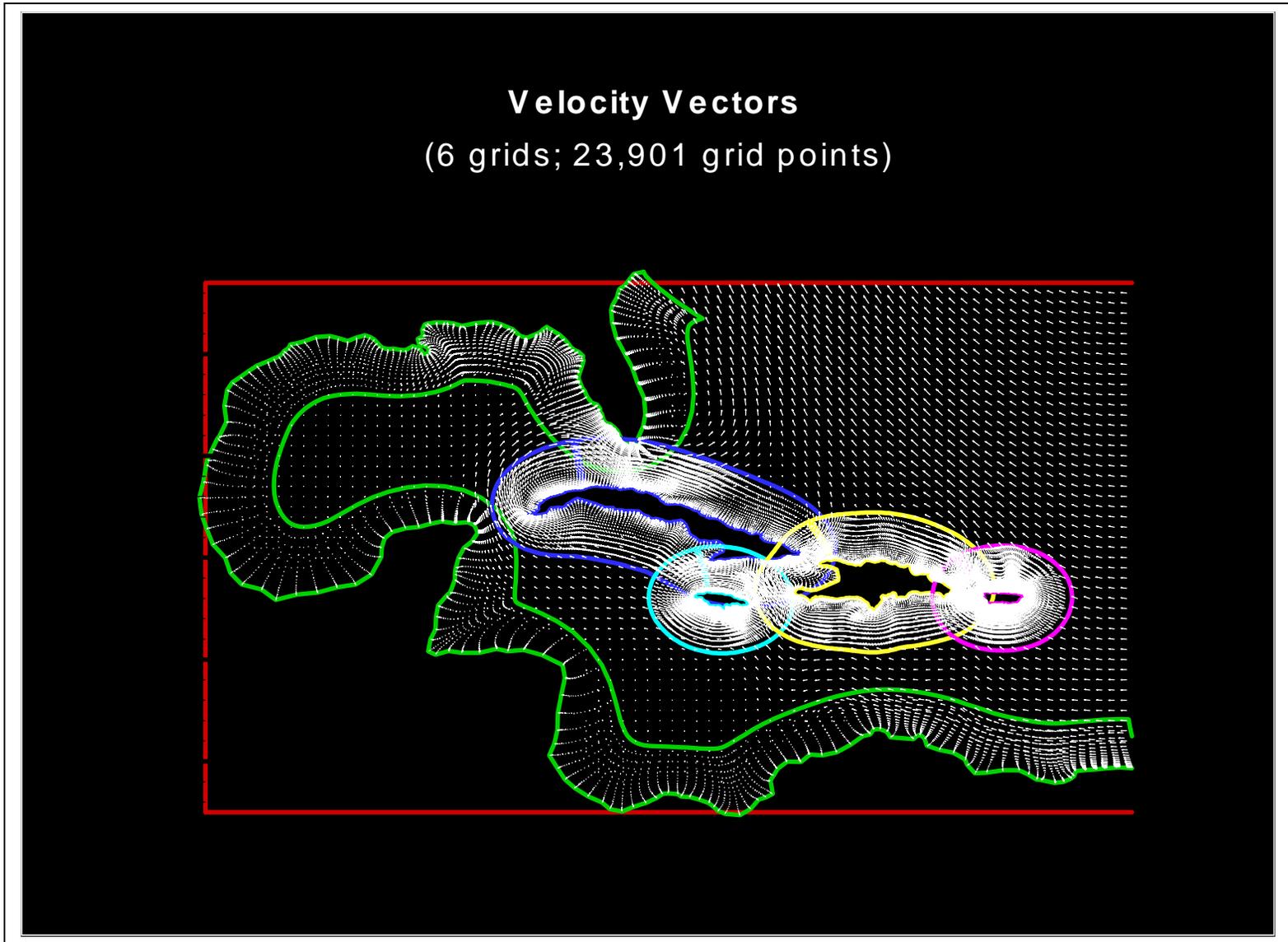
b) 32 subgrids generated by the BREAKUP code

Figure 8. BREAKUP subgrids for an F16's forward fuselage and intake duct .



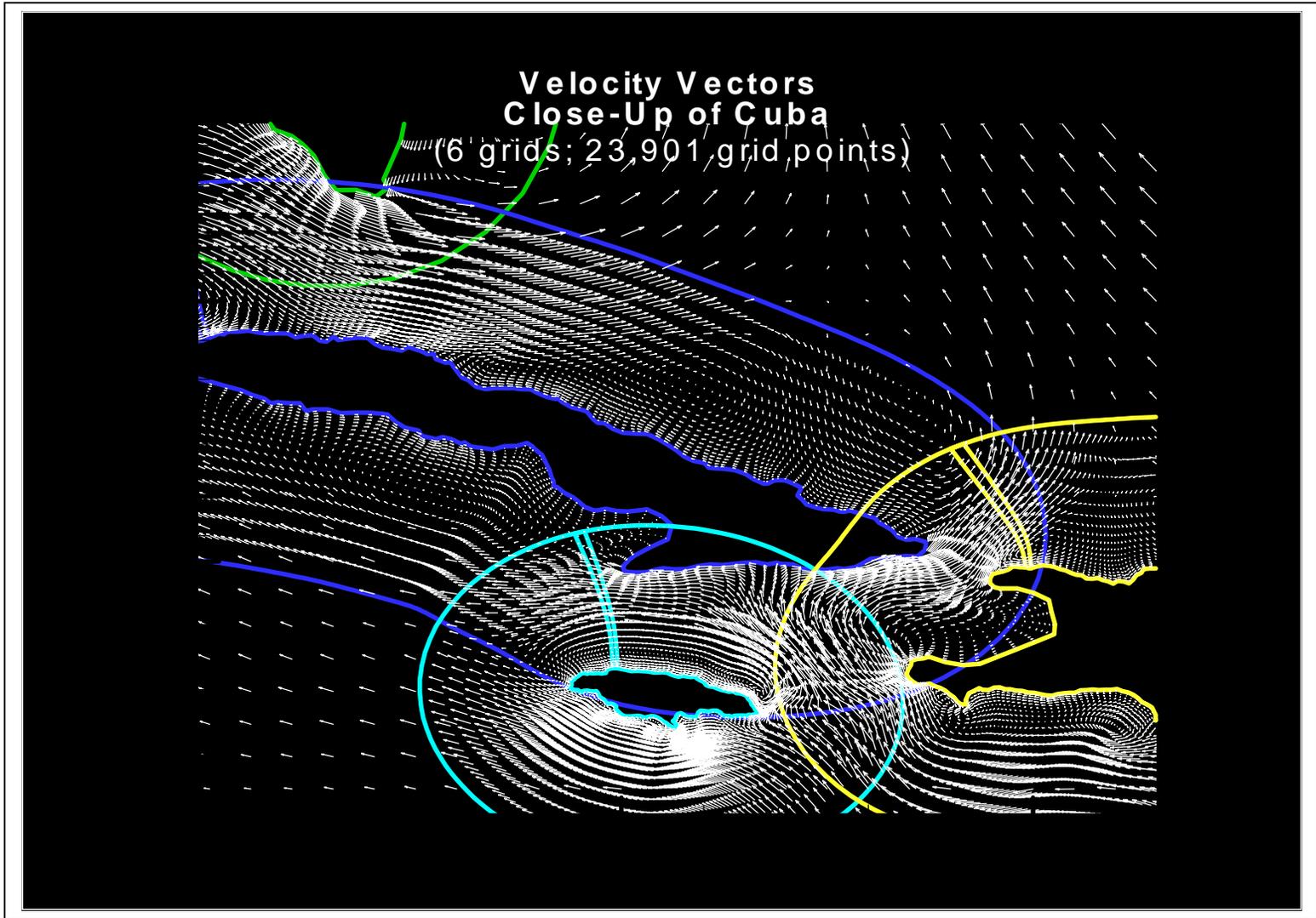
a) Grids

Figure 9. Original six overset grids for the Gulf of Mexico, with serial solution.



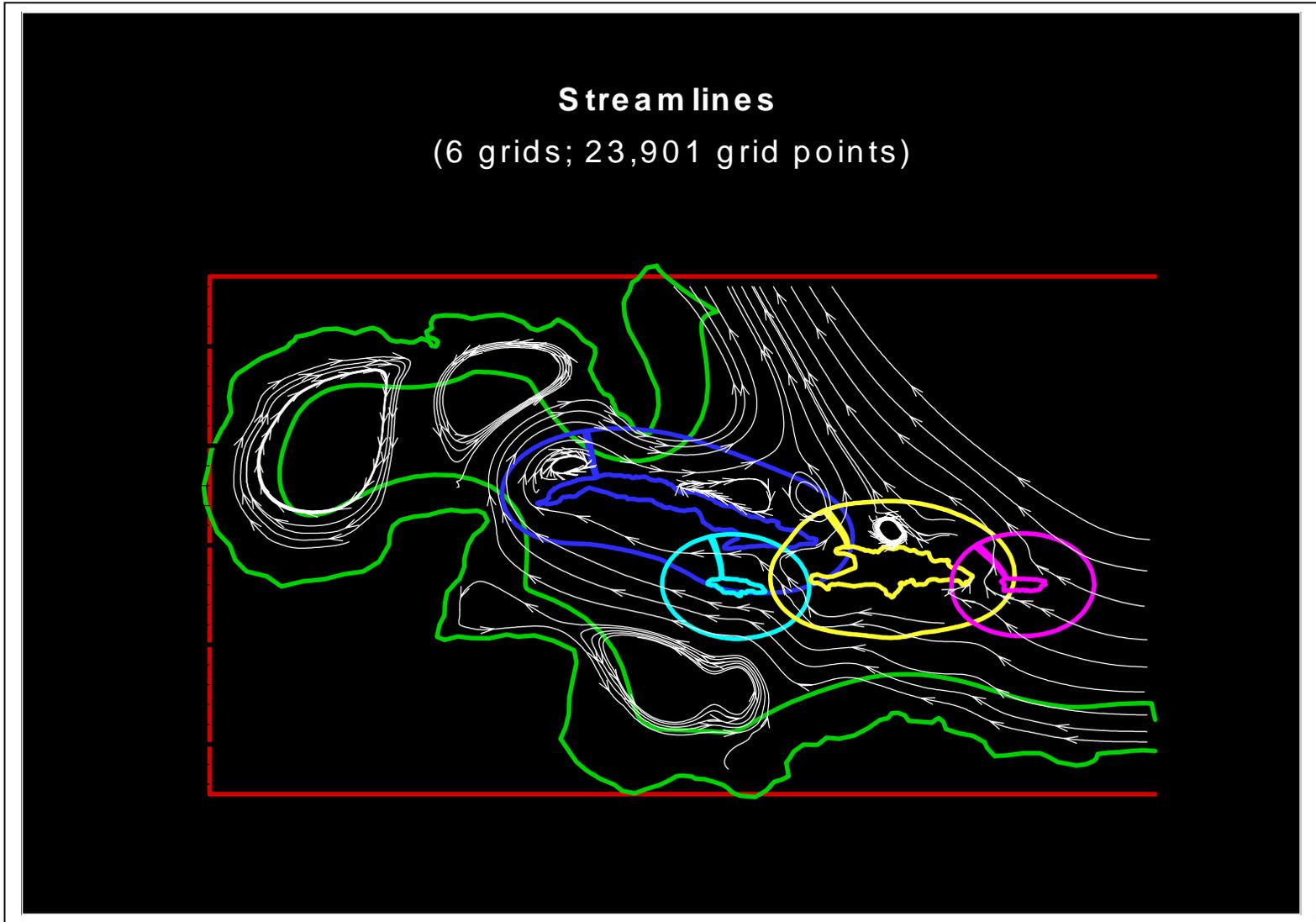
b) Velocity vectors

Figure 9. Continued.



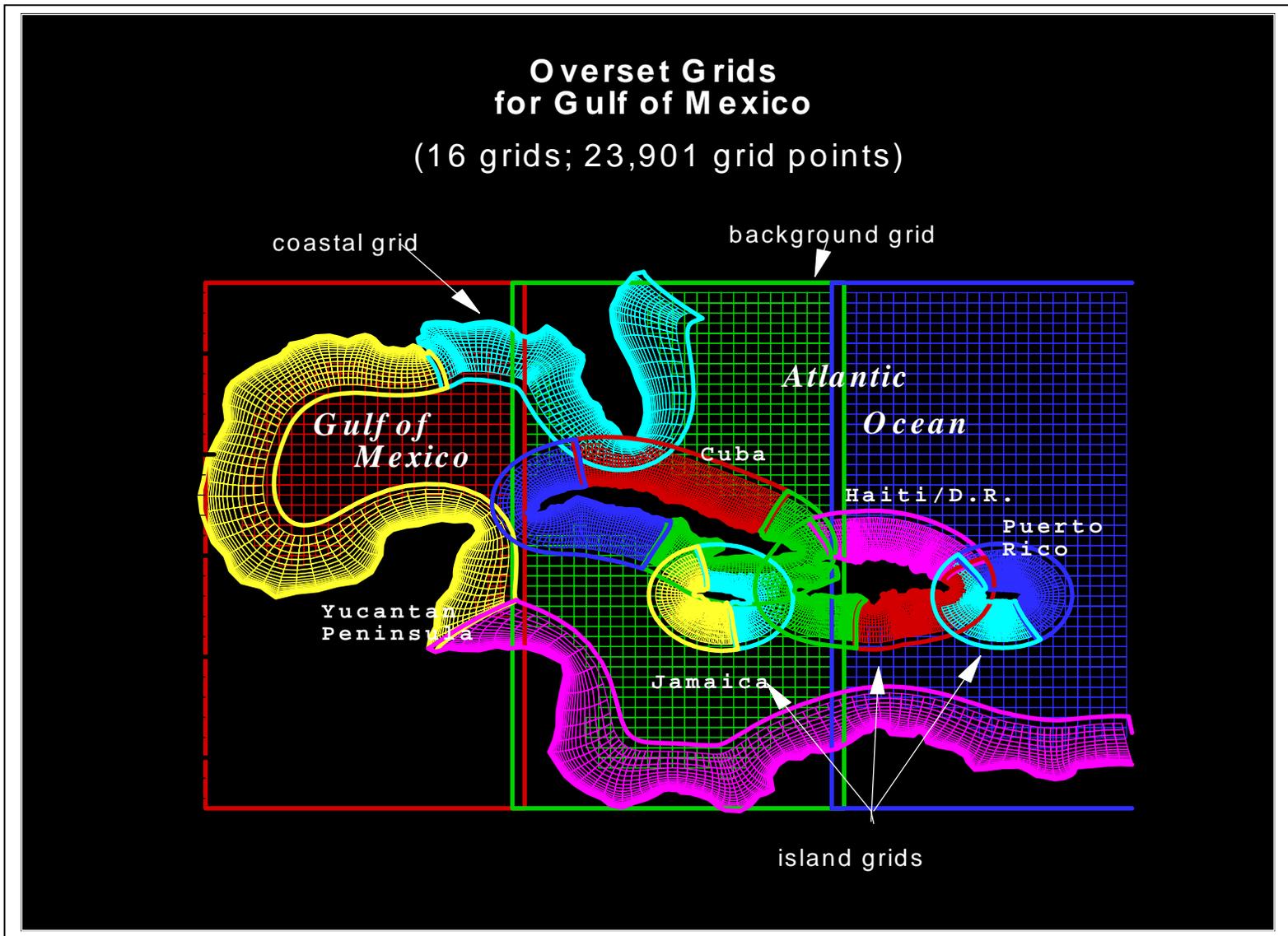
c) Close-up of velocity vectors around Cuba

Figure 9. Continued.



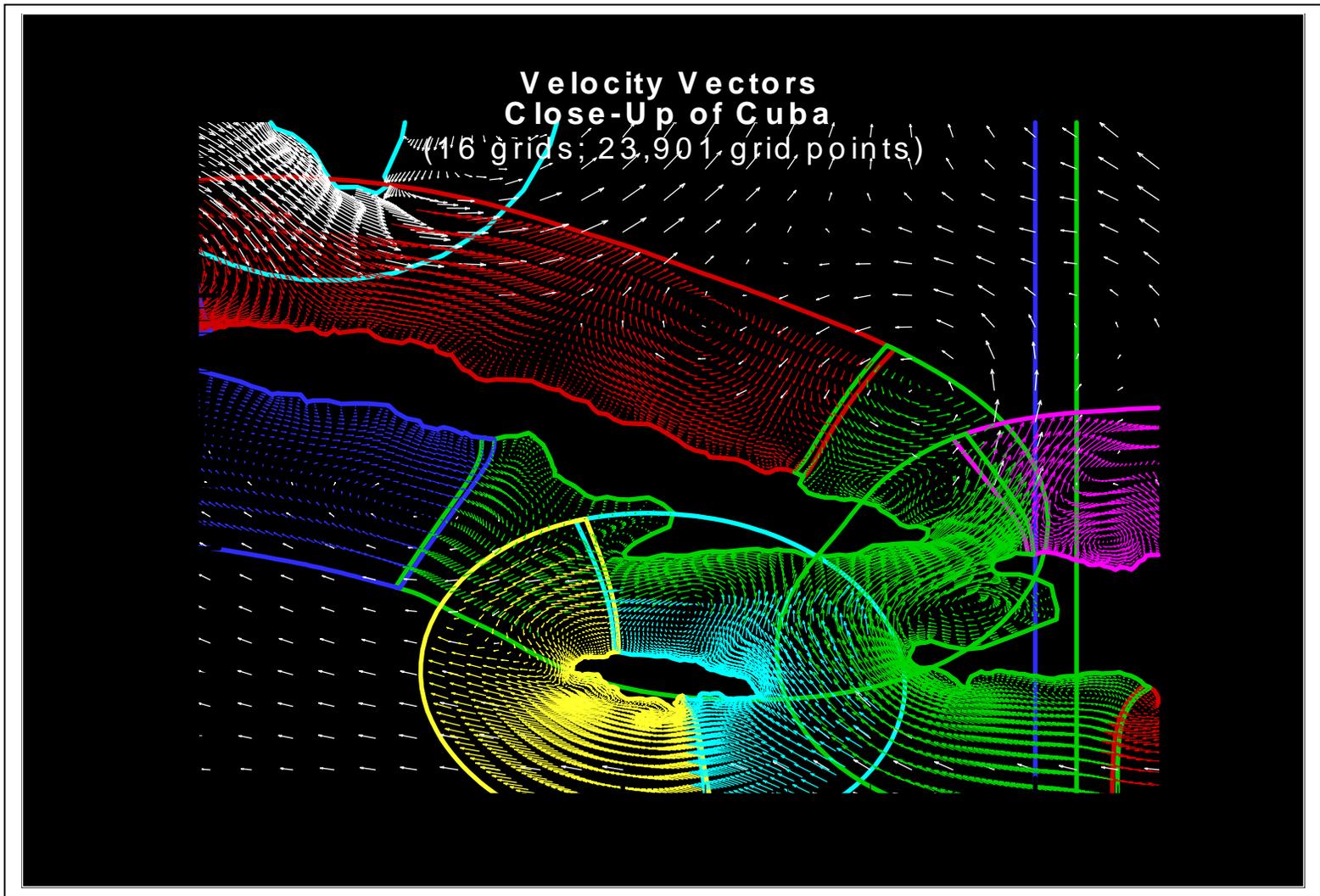
d) Streamlines

Figure 9. Concluded



a) Grids

Figure 10. Sixteen BREAKUP-generated overset grids for the Gulf of Mexico.



b) Close-up of velocity vectors around Cuba

Figure 10. Concluded.

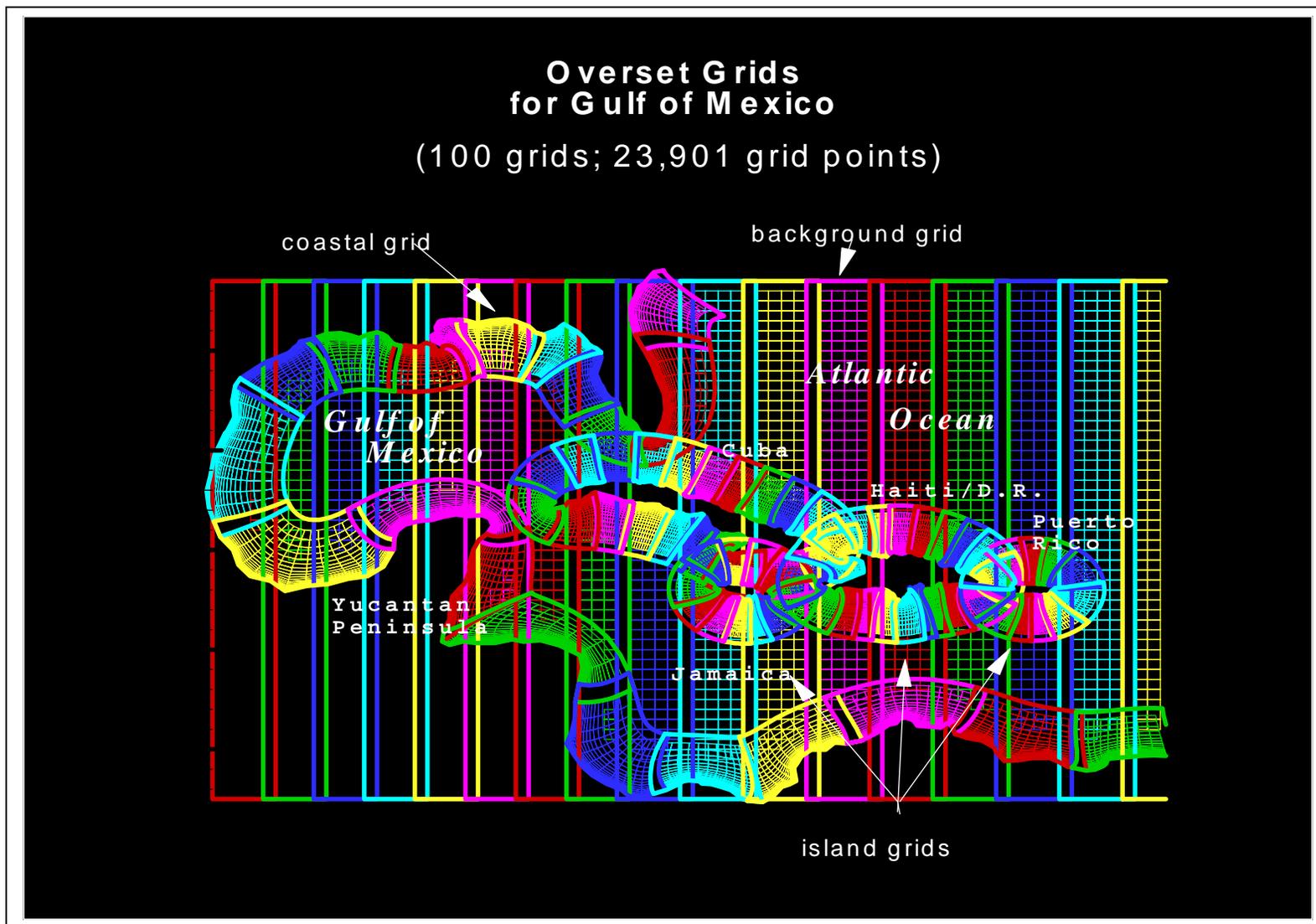
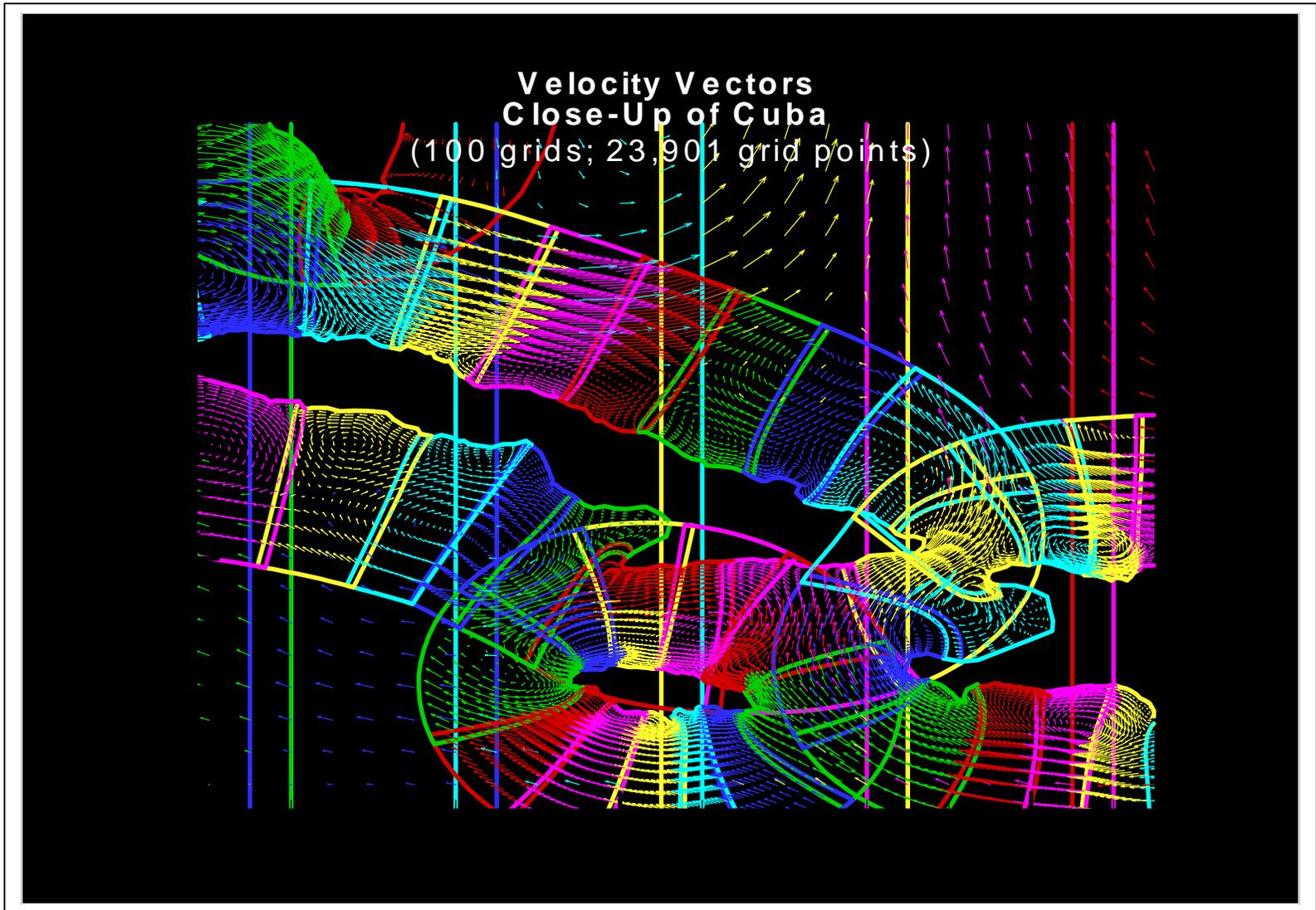


Figure 11. One hundred BREAKUP-generated overset subgrids for the Gulf of Mexico.



b) Close-up of velocity vectors around Cuba

Figure 11. Concluded.

## Appendix A

### BREAKUP Subroutines and Their Description

Listed below in alphabetical order are the subroutines comprising BREAKUP. A brief description is given of each, along with the list of calling routines and the routines that are called.

#### 1. **average**

Purpose:

This subroutine takes in all grids and computes average number of grid points per grid.

Calling routines:

breakup\_for\_fun  
breakup\_grids  
chooser

Called routines, in order of appearance:

NONE

#### 2. **base\_search**

Purpose:

This subroutine searches for base points for overset Chimera-type grids

Calling routines:

main

Called routines, in order of appearance:

get\_zone  
get\_zone

#### 3. **break**

Purpose:

This subroutine breaks up grids into a user-specified number of zones.

Calling routines:

breakup\_for\_fun  
breakup\_grids  
chooser

Called routines, in order of appearance:

min\_surf\_vol\_ratio  
subgrid\_dimensions  
grid\_point\_comparison  
global\_indices  
get\_global\_index  
write\_subgrids

get\_global\_index  
write\_subgrids  
out1planetoproc\_break  
outallplanestoproc\_break  
out1planetoplt3d\_break  
outallplanestoplt3d\_break

#### **4. breakup\_for\_fun**

Purpose:

Main calling code for breaking up grids without PEGSUS interpolation coefficients and without enforcing overlapped regions.

Calling routines:

MAIN

Called routines, in order of appearance:

min\_surf\_vol\_ratio  
subgrid\_dimensions  
grid\_point\_comparison  
global\_indices  
write\_subgrids  
read\_grid\_header  
average  
break

#### **5. breakup\_grids**

Purpose:

Main calling code for breaking up grids without PEGSUS interpolation coefficients; overlapped regions are generated between subgrids.

Calling routines:

MAIN

Called routines, in order of appearance:

read\_grid\_header  
average  
break

#### **6. chooser**

Purpose:

This subroutine gives the user the option to break up grids for parallel processing.

Calling routines:

MAIN

Called routines, in order of appearance:

average  
break  
link\_overlap  
link\_overset  
nobreak

## 7. decimate

Purpose:

This subroutine reads data from file >3D\_OVERSET\_TABLE< (unit 15) and constructs file >2D\_OVERSET\_TABLE< by searching on the target jkl's from file >3D\_OVERSET\_TABLE< and extracting only those with a to-be-calculated target index.

Calling routines:

link\_overset  
nobreak

Called routines, in order of appearance:

NONE

## 8. footer

Purpose:

This subroutine signs off.

Calling routines:

MAIN

Called routines, in order of appearance:

fdate

## 9. get\_global\_index

Purpose:

This subroutine is called to give the index range for subgrids using global indices. It is called from various locations throughout the code.

Calling routines:

break  
link\_overset  
out1planetoplt3d\_break  
out1planetoproc\_break  
outallplanetoplt3d\_break  
outallplanetoproc\_break  
write\_subgrids

Called routines, in order of appearance:

NONE

## **10. get\_zone**

Purpose:

This subroutine reads in the proper zone 'nz' from the grid file in any order

Calling routines:

base\_search  
link\_overset  
nobreak  
target\_search

Called routines, in order of appearance:

NONE

## **11. global\_indices**

Purpose:

This subroutine sets up the global indices needed to break up the zone. Call this routine after calling subroutine subgrid\_dimensions.

Calling routines:

break  
breakup\_for\_fun

Called routines, in order of appearance:

NONE

## **12. grid\_point\_comparison**

Purpose:

This subroutine calculates the total number of points for the original grid, and the total number of grid points for the subgrids of the original grid, and compares the two. The program stops if there is a difference. This is to provide some error checking.

Calling routines:

break  
breakup\_for\_fun

Called routines, in order of appearance:

NONE

## **13. header**

Purpose:

This subroutine prints out the header for output.

Calling routines:

MAIN

Called routines, in order of appearance:

fdate

#### **14. indexx**

Purpose:

Subroutine INDEXX indexes an array `iarr(1:n)`, *i.e.*, outputs the array `indx(1:n)` such that `iarr(indx(j))` is in ascending order for  $j=1,2,\dots,N$ . The input quantities `n` and `iarr` are not changed.

Calling routines:

`sort`

Called routines, in order of appearance:

NONE

#### **15. info**

Purpose:

This subroutine lists variables of interest, filenames, *etc.*, to aid the user in understanding the program.

Calling routines:

NONE

Called routines, in order of appearance:

NONE

#### **16. link\_overlap**

Purpose:

This subroutine links overlapped (patched) subgrids in a zone. This subroutine is not called for overlapped regions for which interpolation coefficients have been generated. The subgrids are assumed to align point-to-point.

Calling routines:

`chooser`

Called routines, in order of appearance:

`write_unravel`

`patch_2d`

#### **17. link\_overset**

Purpose:

This subroutine links grid subgrids for overlapped embedded grids.

Calling routines:

`chooser`

Called routines, in order of appearance:

`get_zone`

`get_global_index`

get\_global\_index  
get\_zone  
sort  
decimate

### **18. min\_surf\_vol\_ratio**

Purpose:

This subroutine calculates the 3-factors for a given number of subgrids, then calculates the surface to volume ratio for each set of 3-factors given the maximum j,k,l dimensions of each grid.

Calling routines:

break  
breakup\_for\_fun

Called routines, in order of appearance:

NONE

### **19. nobreak**

Purpose:

This subroutine keeps all grids as was output by the PEGSUS code. File 3D\_OVERSET\_TABLE is generated listing all base and target grid points. Grids are output as one file or as multiple files, user choice. Essentially, output is for running a flow solver on a serial machine or for distributed computing, not for massively parallel applications on concurrent processors.

Calling routines:

chooser

Called routines, in order of appearance:

read\_zone\_header  
read\_zone  
out1planetoproc\_nobreak  
outallplanestoproc\_nobreak  
out1planetoplt3d\_nobreak  
outallplanestoplt3d\_nobreak  
get\_zone  
get\_zone  
sort  
decimate

### **20. out1planetoplt3d\_break**

Purpose:

The purpose of this subroutine is to write one plane of each subgrid to file GRIDS (UNIT 20) in PLOT3D format for graphics.

Calling routines:

Break

Called routines, in order of appearance:

read\_zone\_header  
read\_zone  
get\_global\_index  
get\_global\_index  
get\_global\_index

### **21. out1planetoplt3d\_nobreak**

Purpose:

The purpose of this subroutine is to write one plane of each of the original grids to file GRIDS2D (UNIT 20) in PLOT3D format for graphics.

Calling routines:

nobreak

Called routines, in order of appearance:

read\_zone\_header  
read\_zone

### **22. out1planetoproc\_break**

Purpose:

The purpose of this subroutine is to output one plane of data for each processor on a massively parallel computer. Output is to file 2D\_\_\_\_.FMT, where \_\_\_\_ is 0000, 0001, 0002, *etc.*

Calling routines:

break

Called routines, in order of appearance:

read\_zone\_header  
read\_zone  
read\_intout  
read\_ibplot  
get\_global\_index  
get\_global\_index  
get\_global\_index

### **23. out1planetoproc\_nobreak**

Purpose:

The purpose of this subroutine is to output one plane of data for each processor on a massively parallel computer. Output is to file 2D\_\_\_\_.FMT, where \_\_\_\_ is 000, 001, 002, *etc.*

Calling routines:

nobreak

Called routines, in order of appearance:

read\_zone\_header

read\_zone  
read\_intout  
read\_ibplot

#### **24. outallplanestopt3d\_break**

Purpose:

The purpose of this subroutine is to write all planes of each subgrid to file GRIDS3D (UNIT 20) in PLOT3D format for graphics.

Calling routines:

break

Called routines, in order of appearance:

read\_zone\_header  
read\_zone  
get\_global\_index

#### **25. outallplanestopt3d\_nobreak**

Purpose:

The purpose of this subroutine is to write all planes of each subgrid to file GRIDS3D (UNIT 20) in PLOT3D format for graphics.

Calling routines:

nobreak

Called routines, in order of appearance:

read\_zone\_header  
read\_zone

#### **26. outallplanestoproc\_break**

Purpose:

The purpose of this subroutine is to output all planes of data for each processor on a massively parallel computer. Output is to file 3D\_\_\_\_.FMT, where \_\_\_\_ is 0000, 0001, 0002, *etc.*

Calling routines:

break

Called routines, in order of appearance:

read\_zone\_header  
read\_zone  
read\_intout  
read\_ibplot  
get\_global\_index

#### **27. outallplanestoproc\_nobreak**

Purpose:

The purpose of this subroutine is to output all planes of data

for each processor on a massively parallel computer. Output is to file XY\_\_\_\_.FMT, where \_\_\_\_ is 0000, 0001, 0002, *etc.*

Calling routines:

nobreak

Called routines, in order of appearance:

read\_zone\_header

read\_zone

read\_intout

read\_ibplot

## **28. patch\_2d**

Purpose:

Reduces the table of 3D patched, *i.e.* point-to-point overlapped grids, to a 2D table for 2D flow solvers.

Calling routines:

link\_overlap

Called routines, in order of appearance:

NONE

## **29. read\_compout**

Purpose:

This subroutine reads data from file COMPOUT, as generated by PEGSUS code.

Calling routines:

MAIN

Called routines, in order of appearance:

read\_ibplot

write\_grids

## **30. read\_grid\_header**

Purpose:

This routine attempts to find the grid file, a plot3d 3D type file, and determines whether it is formatted or unformatted, single or multiple zone, and with or without an iblank array. It reads in the dimensions of the grid and performs parameter checks, and leaves the grid file opened with the pointer after the records containing the dimensions.

Much of this subroutine taken from code written by Phil Stuart of NASA JSC.

Calling routines:

breakup\_for\_fun

breakup\_grids

Called routines, in order of appearance:

NONE

### **31. read\_ibplot**

Purpose:

This subroutine reads the IBPLOT array, as generated by PEGSUS code, one zone at a time

Calling routines:

MAIN  
out1planetoproc\_break  
out1planetoproc\_nobreak  
outallplanestoproc\_break  
outallplanestoproc\_nobreak  
read\_compout

Called routines, in order of appearance:

NONE

### **32. read\_intout**

Purpose:

This subroutine reads the INTOUT file, as generated by the PEGSUS code, one zone at a time. It includes the IBLANK PEGSUS code array.

Calling routines:

MAIN  
out1planetoproc\_nobreak  
outallplanestoproc\_break  
outallplanestoproc\_nobreak

Called routines, in order of appearance:

exit  
exit

### **33. read\_zone**

Purpose:

This subroutine sequentially reads the grid file one zone at a time. The grid file read in by this subroutine is assumed to be a 3-d PLOT3D formatted multi-zone file. The read pointer has been positioned past the header data by previously calling subroutine read\_zone\_header.

Calling routines:

nobreak  
out1planetoplt3d\_break  
out1planetoplt3d\_nobreak  
out1planetoproc\_break  
out1planetoproc\_nobreak  
outallplanestoplt3d\_break

outallplanestoplt3d\_nobreak  
outallplanestoproc\_break  
outallplanestoproc\_nobreak

Called routines, in order of appearance:  
NONE

### **34. read\_zone\_header**

Purpose:

This subroutine reads the grid file header and leaves the read pointer at the start of the grid points for the first zone. The grid file read in by this subroutine is assumed to be a 3-d PLOT3D formatted multi-zone file.

Calling routines:

nobreak  
out1planetoplt3d\_break  
out1planetoplt3d\_nobreak  
out1planetoproc\_break  
out1planetoproc\_nobreak  
outallplanestoplt3d\_break  
outallplanestoplt3d\_nobreak  
outallplanestoproc\_break  
outallplanestoproc\_nobreak

Called routines, in order of appearance:  
NONE

### **35. sort**

Purpose:

This subroutine reads data from scratch file SCRATCH25, sorts on the base processor, then sorts the target processor for each base processor, so a double sort is needed. Scratch file SCRATCH30 is used as a temporary scratch file between sorts.

Calling routines:

link\_overset  
nobreak

Called routines, in order of appearance:  
indexx  
indexx

### **36. subgrid\_dimensions**

Purpose:

This subroutine calculates the dimensions of the subgrids.

Calling routines:

break

breakup\_for\_fun  
Called routines, in order of appearance:  
NONE

### **37. target\_search**

Purpose:

This subroutine searches for target points for overset Chimera-type grids

Calling routines:

MAIN

Called routines, in order of appearance:

get\_zone

get\_zone

### **38. write\_base\_target\_points**

Purpose:

This subroutine writes base (stencil) and target (boundary) points for each zone in PLOT3D format.

Calling routines:

MAIN

Called routines, in order of appearance:

NONE

### **39. write\_grids**

Purpose:

This subroutine writes grids, read from PEGUS output, in PLOT3D format. The data in this format is read in later and subdivided into the user-specified number of subgrids.

Calling routines:

read\_compout

Called routines, in order of appearance:

NONE

### **40. write\_subgrids**

Purpose:

This subroutine writes out the subgrids formed in subroutine SUBGRID\_DIMENSIONS and subroutine GLOBAL\_INDICES.

Calling routines:

break

breakup\_for\_fun

Called routines, in order of appearance:

get\_global\_index

#### **41. write\_unravel**

Purpose:

This subroutine outputs to file UNRAVEL. Output depends on iflag.

Calling routines:

MAIN

link\_overlap

Called routines, in order of appearance:

NONE

## Appendix B

### List of Variables used in BREAKUP

Listed below are the more significant variables used in BREAKUP. A brief description is given of each. If not listed below, the variable's meaning is probably obvious from the context of the coding.

Name	Description
j_subgrid(index,nz) k_subgrid(index,nz) l_subgrid(index,nz)	jmax,kmax,lmax of each subgrid, where index=(j-1)*idimk(nz)*idiml(nz)+ (k-1)*idiml(nz)+1
idimj(nz) idimk(nz) idiml(nz)	number of subgrids for each overset grid
jindex_global(index,nz) kindex_global(index,nz) lindex_global(index,nz)	indices for each subgrid based on total dimension of the grid in which the subgrid resides; index=(j-1)*idimk(nz)*idiml(nz)+ (k-1)*idiml(nz)+1
x(index) y(index) z(index)	x, y, and z for each zone nz. Note that if the zone is dimensioned jmax,kmax,lmax, then index=(j-1)*kmax*lmax+(k-1)*lmax+1, where j = 1 to jmax, k = 1 to kmax, l = 1 to lmax.
xb(i,nz) yb(i,nz) zb(i,nz)	locations for base points used in interpolation stencil
jmax_compout(nz) kmax_compout(nz) lmax_compout(nz)	indices from PEGSUS; dimensions for each zone, where (zone=1,nz)
jmax_intout(nz) kmax_intout(nz) lmax_intout(nz)	indices from PEGSUS code for interpolation coefficients for each mesh
ibpnts(nz)	number of interpolation boundary points per mesh
iipnts(nz)	number of interpolation stencils/mesh
ieptr(nz)	end pointer for interpolation stencil list per mesh
iisptr(nz)	starting pointer for interpolation stencil list per mesh
ji(i) ki(i) li(i)	indices for target points from PEGSUS

Name	Description
-----	-----
jb(i) kb(i) lb(i)	indices for base points from PEGSUS
ibc	interpolation boundary point pointer
ibplot(index)	array read in from PEGSUS file IBPLOT and is the IBLANK array typically used in plotting PLOT3D files
iblack(index)	array read in from PEGSUS file INTOUT; used as a flag to the flow solver to tell it that particular boundary points are updated by PEGSUS interpolation and not by the solver
dx(i) dy(i) dz(i)	interpolation coefficients from PEGSUS

## Appendix C

### Sample Output from BREAKUP (file BREAKUP.OUT)

Note: line numbers have been added for reference

```
1. *****
2. ### Program BREAKUP ###
3. The PARAMETER statement for the current run is set as follows:
4. nzne = 30
5. jmax = 143
6. kmax = 31
7. lmax = 30
8. max_subs = 85
9. You have chosen the following option:
10. -----
11. 2) BREAKUP PLOT3D MULTI-BLOCK GRIDS; OVERLAP
12. SUBGRIDS
13. -----
14. You have one or more grids in a multiblock PLOT3D
15. file that need to be subdivided; no PEGSUS
16. interpolation coefficients are available or
17. needed. BREAKUP will subdivide the original
18. grids into subgrids. The subgrids will be
19. modified to overlap for point-to-point
20. communications.
21. Read in grid. Grid is assumed to be in PLOT3D
22. format. BREAKUP can discern if file is
23. formatted or binary, and if grids have the
24. iblank array. Subgrids will be output to file
25. GRID_DIVIDED.G in formatted PLOT3D format.
26. input grid file must have one of the
27. following names:
28. X.FMT
29. X.DAT
30. XY.FMT
31. XY.DAT
32. XYZ.FMT
33. XYZ.DAT
34. GRID.FMT
35. GRID.DAT
36. G.FMT
37. G.DAT
38. Attempting to open file named: XY.FMT
39. File opened successfully.
40. #      j      k      l      Total Points
41. -----
42. 1      3      77      53      12243
43. 2      3      249      20      14940
44. 3      3      240      20      14400
45. 4      3      135      20      8100
46. 5      3      234      20      14040
47. 6      3      133      20      7980
48. Zone 2 has the largest number of grid points:
49. jmax = 3
50. kmax = 249
51. lmax = 20
52. Total number of grid points for this zone: 14940
53. End of grid-file scan.
54. This is a formatted, multiple zone grid file with an iblank array.
```

55. nz = 1 jmax = 3 kmax = 77 lmax = 53  
 56. nz = 2 jmax = 3 kmax = 249 lmax = 20  
 57. nz = 3 jmax = 3 kmax = 240 lmax = 20  
 58. nz = 4 jmax = 3 kmax = 135 lmax = 20  
 59. nz = 5 jmax = 3 kmax = 234 lmax = 20  
 60. nz = 6 jmax = 3 kmax = 133 lmax = 20

61. Total number of grid points = 71703

62. -----

63. Grid	Jmax	Kmax	Lmax	Total
64. GRID0001.G	3	77	53	12243
65. GRID0002.G	3	249	20	14940
66. GRID0003.G	3	240	20	14400
67. GRID0004.G	3	135	20	8100
68. GRID0005.G	3	234	20	14040
69. GRID0006.G	3	133	20	7980

70. Total number of grid points = 71703

71. Total number of original zones = 6

72. Average number of grid points/grid = 11950.50

73. Parallel processing parameters:

74. Number of nodes to be used: 16

75. Number of grid points in zone 1: 12243  
 76. Number of grid points in zone 2: 14940  
 77. Number of grid points in zone 3: 14400  
 78. Number of grid points in zone 4: 8100  
 79. Number of grid points in zone 5: 14040  
 80. Number of grid points in zone 6: 7980

81. Total number of grid points over all grids: 71703  
 82. Average number of grid points per grid: 11950.5  
 83. Average number of grid points per processor: 4481.0

84. For zone 1 use 3 processors.  
 85. For zone 2 use 3 processors.  
 86. For zone 3 use 3 processors.  
 87. For zone 4 use 2 processors.  
 88. For zone 5 use 3 processors.  
 89. For zone 6 use 2 processors.

90. User-specified nodes to be used = 16  
 91. Nodes calculated to be used = 16

92. Calculate j\*k\*l values for subgrids for minimum  
 93. area-to-volume ratio.

94. =====

95. For mesh GRID0001.G:

96. List all possible factors for number of subgrids  
 97. for this zone.

98. ITAG	J	K	L
99. ----	---	---	---
100. 1	1	1	3
101. 2	1	3	1
102. 3	3	1	1

103. Finding min(surf/vol) for:  
 104. Jmax = 3  
 105. Kmax = 77  
 106. Lmax = 53

107. itag	surf/vol	min(surf/vol)
108. 1	0.805848	0.805848

```

109. 2      0.782325      0.782325
110. 3      2.063710      0.782325

111. Final values for min(surf/vol) for this zone are:
112. itag = 2
113. min(surf/vol) = 0.782325
114. N/Jmax = 1 / 3 ==> approx. 3 pts/subgrid in J direction
115. M/Kmax = 3 / 77 ==> approx. 26 pts/subgrid in K direction
116. P/Lmax = 1 / 53 ==> approx. 53 pts/subgrid in L direction

117. Warning: Kmaxx/M not evenly divisible!
118. K-remainder = 1

119. BREAKUP will make adjustments in the size of the
120. subgrids to compensate for the above remainders.
121. Adjustments will start at the boundaries and
122. progress inward. Hence, when remainders appear,
123. outer subgrids will have their dimensions
124. increased in the corresponding direction.

125. Final subgrid dimensions are:

126. Grid #      Jcube      Kcube      Lcube      Jmax      Kmax      Lmax      Total
127. -----      -
128. 1          1          1          1          3          27          53          4293
129. 2          1          2          1          3          26          53          4134
130. 3          1          3          1          3          26          53          4134

131. Compare total number of grid points with original
132. in this zone:
133. Subtracted common faces in K-direction.

134. Original no. of points in this zone = 12243
135. Calculated no. of points in this zone = 12243

136. =====
137. For mesh GRID0002.G:

138. List all possible factors for number of subgrids
139. for this zone.

140. ITAG      J      K      L
141. ----      -      -      -
142. 1          1      1      3
143. 2          1      3      1
144. 3          3      1      1

145. Finding min(surf/vol) for:
146. Jmax = 3
147. Kmax = 249
148. Lmax = 20

149. itag      surf/vol      min(surf/vol)
150. 1          0.974699      0.974699
151. 2          0.790763      0.790763
152. 3          2.108032      0.790763

153. Final values for min(surf/vol) for this zone are:
154. itag = 2
155. min(surf/vol) = 0.790763
156. N/Jmax = 1 / 3 ==> approx. 3 pts/subgrid in J direction
157. M/Kmax = 3 / 249 ==> approx. 83 pts/subgrid in K direction
158. P/Lmax = 1 / 20 ==> approx. 20 pts/subgrid in L direction

159. Warning: Kmaxx/M not evenly divisible!
160. K-remainder = 2

161. BREAKUP will make adjustments in the size of the
162. subgrids to compensate for the above remainders.
163. Adjustments will start at the boundaries and
164. progress inward. Hence, when remainders appear,
165. outer subgrids will have their dimensions
166. increased in the corresponding direction.

167. Final subgrid dimensions are:

168. Grid #      Jcube      Kcube      Lcube      Jmax      Kmax      Lmax      Total
169. -----      -
170. 1          1          1          1          3          84          20          5040

```

```

171.      2          1          2          1          3          83          20          4980
172.      3          1          3          1          3          84          20          5040

173.      Compare total number of grid points with original
174.      in this zone:
175.      Subtracted common faces in K-direction.

176.      Original no. of points in this zone =      14940
177.      Calculated no. of points in this zone =      14940

178.      =====
179.      For mesh GRID0003.G:

180.      List all possible factors for number of subgrids
181.      for this zone.

182.      ITAG          J          K          L
183.      ----          - - -          - - -          - - -
184.      1             1             1             3
185.      2             1             3             1
186.      3             3             1             1

187.      Finding min(surf/vol) for:
188.      Jmax =          3
189.      Kmax =          240
190.      Lmax =          20

191.      itag          surf/vol          min(surf/vol)
192.      1             0.975000          0.975000
193.      2             0.791667          0.791667
194.      3             2.108333          0.791667

195.      Final values for min(surf/vol) for this zone are:
196.      itag =          2
197.      min(surf/vol) =      0.791667
198.      N/Jmax = 1 / 3 ==> approx. 3 pts/subgrid in J direction
199.      M/Kmax = 3 / 240 ==> approx. 80 pts/subgrid in K direction
200.      P/Lmax = 1 / 20 ==> approx. 20 pts/subgrid in L direction

201.      Warning: Kmaxx/M not evenly divisible!
202.      K-remainder = 2

203.      BREAKUP will make adjustments in the size of the
204.      subgrids to compensate for the above remainders.
205.      Adjustments will start at the boundaries and
206.      progress inward. Hence, when remainders appear,
207.      outer subgrids will have their dimensions
208.      increased in the corresponding direction.

209.      Final subgrid dimensions are:

210.      Grid #          Jcube          Kcube          Lcube          Jmax          Kmax          Lmax          Total
211.      -----          - - -          - - -          - - -          - - -          - - -          - - -          - - -
212.      1             1             1             1             3             81             20             4860
213.      2             1             2             1             3             80             20             4800
214.      3             1             3             1             3             81             20             4860

215.      Compare total number of grid points with original
216.      in this zone:
217.      Subtracted common faces in K-direction.

218.      Original no. of points in this zone =      14400
219.      Calculated no. of points in this zone =      14400

220.      =====
221.      For mesh GRID0004.G:

222.      List all possible factors for number of subgrids
223.      for this zone.

224.      ITAG          J          K          L
225.      ----          - - -          - - -          - - -
226.      1             1             1             2
227.      2             1             2             1
228.      3             2             1             1

229.      Finding min(surf/vol) for:

```

```

230. Jmax = 3
231. Kmax = 135
232. Lmax = 20

233. itag surf/vol min(surf/vol)
234. 1 0.881481 0.881481
235. 2 0.796296 0.796296
236. 3 1.448148 0.796296

237. Final values for min(surf/vol) for this zone are:
238. itag = 2
239. min(surf/vol) = 0.796296
240. N/Jmax = 1 / 3 ==> approx. 3 pts/subgrid in J direction
241. M/Kmax = 2 / 135 ==> approx. 68 pts/subgrid in K direction
242. P/Lmax = 1 / 20 ==> approx. 20 pts/subgrid in L direction

243. Final subgrid dimensions are:

244. Grid # Jcube Kcube Lcube Jmax Kmax Lmax Total
245. -----
246. 1 1 1 1 3 68 20 4080
247. 2 1 2 1 3 68 20 4080

248. Compare total number of grid points with original
249. in this zone:
250. Subtracted common faces in K-direction.

251. Original no. of points in this zone = 8100
252. Calculated no. of points in this zone = 8100

253. =====
254. For mesh GRID0005.G:

255. List all possible factors for number of subgrids
256. for this zone.

257. ITAG J K L
258. ---- - - - -
259. 1 1 1 3
260. 2 1 3 1
261. 3 3 1 1

262. Finding min(surf/vol) for:
263. Jmax = 3
264. Kmax = 234
265. Lmax = 20

266. itag surf/vol min(surf/vol)
267. 1 0.975214 0.975214
268. 2 0.792308 0.792308
269. 3 2.108547 0.792308

270. Final values for min(surf/vol) for this zone are:
271. itag = 2
272. min(surf/vol) = 0.792308
273. N/Jmax = 1 / 3 ==> approx. 3 pts/subgrid in J direction
274. M/Kmax = 3 / 234 ==> approx. 78 pts/subgrid in K direction
275. P/Lmax = 1 / 20 ==> approx. 20 pts/subgrid in L direction

276. Warning: Kmaxx/M not evenly divisible!
277. K-remainder = 2

278. BREAKUP will make adjustments in the size of the
279. subgrids to compensate for the above remainders.
280. Adjustments will start at the boundaries and
281. progress inward. Hence, when remainders appear,
282. outer subgrids will have their dimensions
283. increased in the corresponding direction.

284. Final subgrid dimensions are:

285. Grid # Jcube Kcube Lcube Jmax Kmax Lmax Total
286. -----
287. 1 1 1 1 3 79 20 4740
288. 2 1 2 1 3 78 20 4680
289. 3 1 3 1 3 79 20 4740

290. Compare total number of grid points with original

```

```

291.   in this zone:
292.   Subtracted common faces in K-direction.

293.   Original no. of points in this zone =      14040
294.   Calculated no. of points in this zone =      14040

295.   =====
296.   For mesh GRID0006.G:

297.   List all possible factors for number of subgrids
298.   for this zone.

299.   ITAG      J      K      L
300.   ----      -      -      -
301.   1          1      1      2
302.   2          1      2      1
303.   3          2      1      1

304.   Finding min(surf/vol) for:
305.   Jmax =          3
306.   Kmax =         133
307.   Lmax =          20

308.   itag      surf/vol      min(surf/vol)
309.   1          0.881704      0.881704
310.   2          0.796742      0.796742
311.   3          1.448371      0.796742

312.   Final values for min(surf/vol) for this zone are:
313.   itag =          2
314.   min(surf/vol) =    0.796742
315.   N/Jmax =    1 /    3 ==> approx.    3 pts/subgrid in J direction
316.   M/Kmax =    2 / 133 ==> approx.   67 pts/subgrid in K direction
317.   P/Lmax =    1 /   20 ==> approx.   20 pts/subgrid in L direction

318.   Final subgrid dimensions are:

319.   Grid #      Jcube      Kcube      Lcube      Jmax      Kmax      Lmax      Total
320.   -----      -      -      -      -      -      -      -
321.   1          1          1          1          3          67          20          4020
322.   2          1          2          1          3          67          20          4020

323.   Compare total number of grid points with original
324.   in this zone:
325.   Subtracted common faces in K-direction.

326.   Original no. of points in this zone =      7980
327.   Calculated no. of points in this zone =      7980

328.   =====
329.   Local and Global Indices for Each Subgrid
330.   =====

331.   Zone =    1, Meshname = GRID0001.G
332.   Grd# Jcube Kcube Lcube Jmax Kmax Lmax Jming Jmaxg Kming Kmaxg Lming Lmaxg
333.   ---- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
334.   1     1     1     1     3    27    53     1     3     1    27     1    53
335.   2     1     2     1     3    26    53     1     3    27    52     1    53
336.   3     1     3     1     3    26    53     1     3    52    77     1    53

337.   Zone =    2, Meshname = GRID0002.G
338.   Grd# Jcube Kcube Lcube Jmax Kmax Lmax Jming Jmaxg Kming Kmaxg Lming Lmaxg
339.   ---- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
340.   1     1     1     1     3    84    20     1     3     1    84     1    20
341.   2     1     2     1     3    83    20     1     3    84   166     1    20
342.   3     1     3     1     3    84    20     1     3   166   249     1    20

343.   Zone =    3, Meshname = GRID0003.G
344.   Grd# Jcube Kcube Lcube Jmax Kmax Lmax Jming Jmaxg Kming Kmaxg Lming Lmaxg
345.   ---- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
346.   1     1     1     1     3    81    20     1     3     1    81     1    20
347.   2     1     2     1     3    80    20     1     3    81   160     1    20
348.   3     1     3     1     3    81    20     1     3   160   240     1    20

```

```

349. Zone = 4, Meshname = GRID0004.G
350. Grd# Jcube Kcube Lcube Jmax Kmax Lmax Jming Jmaxg Kming Kmaxg Lming Lmaxg
351. -----
352. 1 1 1 1 3 68 20 1 3 1 68 1 20
353. 2 1 2 1 3 68 20 1 3 68 135 1 20

```

```

354. Zone = 5, Meshname = GRID0005.G
355. Grd# Jcube Kcube Lcube Jmax Kmax Lmax Jming Jmaxg Kming Kmaxg Lming Lmaxg
356. -----
357. 1 1 1 1 3 79 20 1 3 1 79 1 20
358. 2 1 2 1 3 78 20 1 3 79 156 1 20
359. 3 1 3 1 3 79 20 1 3 156 234 1 20

```

```

360. Zone = 6, Meshname = GRID0006.G
361. Grd# Jcube Kcube Lcube Jmax Kmax Lmax Jming Jmaxg Kming Kmaxg Lming Lmaxg
362. -----
363. 1 1 1 1 3 67 20 1 3 1 67 1 20
364. 2 1 2 1 3 67 20 1 3 67 133 1 20

```

365. Breaking up grids into subgrids WITH NO overlap:

```

366. Zone Subzone Prc# Jmax Kmax Lmax Points J-rel range K-rel range L-rel range
367. 1 1 1 3 27 53 4293 1 to 3 1 to 27 1 to 53
368. 1 2 2 3 26 53 4134 1 to 3 27 to 52 1 to 53
369. 1 3 3 3 26 53 4134 1 to 3 52 to 77 1 to 53
370. -----
371. 2 1 4 3 84 20 5040 1 to 3 1 to 84 1 to 20
372. 2 2 5 3 83 20 4980 1 to 3 84 to 166 1 to 20
373. 2 3 6 3 84 20 5040 1 to 3 166 to 249 1 to 20
374. -----
375. 3 1 7 3 81 20 4860 1 to 3 1 to 81 1 to 20
376. 3 2 8 3 80 20 4800 1 to 3 81 to 160 1 to 20
377. 3 3 9 3 81 20 4860 1 to 3 160 to 240 1 to 20
378. -----
379. 4 1 10 3 68 20 4080 1 to 3 1 to 68 1 to 20
380. 4 2 11 3 68 20 4080 1 to 3 68 to 135 1 to 20
381. -----
382. 5 1 12 3 79 20 4740 1 to 3 1 to 79 1 to 20
383. 5 2 13 3 78 20 4680 1 to 3 79 to 156 1 to 20
384. 5 3 14 3 79 20 4740 1 to 3 156 to 234 1 to 20
385. -----
386. 6 1 15 3 67 20 4020 1 to 3 1 to 67 1 to 20
387. 6 2 16 3 67 20 4020 1 to 3 67 to 133 1 to 20

```

388. Breaking up grids into subgrids WITH overlap:

```

389. Zone Subzone Prc# Jmax Kmax Lmax Points J-rel range K-rel range L-rel range
390. 1 1 1 3 28 53 4452 1 to 3 1 to 28 1 to 53
391. 1 2 2 3 27 53 4293 1 to 3 27 to 53 1 to 53
392. 1 3 3 3 26 53 4134 1 to 3 52 to 77 1 to 53
393. -----
394. 2 1 4 3 85 20 5100 1 to 3 1 to 85 1 to 20
395. 2 2 5 3 84 20 5040 1 to 3 84 to 167 1 to 20
396. 2 3 6 3 84 20 5040 1 to 3 166 to 249 1 to 20
397. -----
398. 3 1 7 3 82 20 4920 1 to 3 1 to 82 1 to 20
399. 3 2 8 3 81 20 4860 1 to 3 81 to 161 1 to 20
400. 3 3 9 3 81 20 4860 1 to 3 160 to 240 1 to 20
401. -----
402. 4 1 10 3 69 20 4140 1 to 3 1 to 69 1 to 20
403. 4 2 11 3 68 20 4080 1 to 3 68 to 135 1 to 20
404. -----
405. 5 1 12 3 80 20 4800 1 to 3 1 to 80 1 to 20
406. 5 2 13 3 79 20 4740 1 to 3 79 to 157 1 to 20
407. 5 3 14 3 79 20 4740 1 to 3 156 to 234 1 to 20
408. -----
409. 6 1 15 3 68 20 4080 1 to 3 1 to 68 1 to 20
410. 6 2 16 3 67 20 4020 1 to 3 67 to 133 1 to 20

```

411. Writing subgrids to file.

412. Choose PLOT3D-output-format of file for subgrids:

```

413.    formatted
414.    unformatted

415.    Option number          1 chosen.

416.    Choose whether you want the IBLANK array attached:
417.    attach iblank array to subgrids
418.    do NOT attach iblank array to subgrids

419.    Option number          1 chosen.

420.    Output file GRID_DIVIDED.G opened as formatted.

421.    Constructing subgrids.

422.    -----
423.    Reading original grid file, zone #    1
424.    This grid contains IBLANK data which will be read.
425.    -----

426.    Constructing subgrid #    1 of    3 subgrids in zone    1
427.    Total count: subgrid #    1 of   16 subgrids
428.    Grid cells overlap 1 cell widths on grid boundaries.
429.    IBLANK data available and will be written to
430.    this subgrid file.
431.    nz =    1 jcube =    1 kcube =    1 lcube =    1
432.    jp3d_min =    1  jp3d_max =    3
433.    kp3d_min =    1  kp3d_max =   28
434.    lp3d_min =    1  lp3d_max =   53

435.    Constructing subgrid #    2 of    3 subgrids in zone    1
436.    Total count: subgrid #    2 of   16 subgrids
437.    Grid cells overlap 1 cell widths on grid boundaries.
438.    IBLANK data available and will be written to
439.    this subgrid file.
440.    nz =    1 jcube =    1 kcube =    2 lcube =    1
441.    jp3d_min =    1  jp3d_max =    3
442.    kp3d_min =   27  kp3d_max =   53
443.    lp3d_min =    1  lp3d_max =   53

444.    Constructing subgrid #    3 of    3 subgrids in zone    1
445.    Total count: subgrid #    3 of   16 subgrids
446.    Grid cells overlap 1 cell widths on grid boundaries.
447.    IBLANK data available and will be written to
448.    this subgrid file.
449.    nz =    1 jcube =    1 kcube =    3 lcube =    1
450.    jp3d_min =    1  jp3d_max =    3
451.    kp3d_min =   52  kp3d_max =   77
452.    lp3d_min =    1  lp3d_max =   53

453.    -----
454.    Reading original grid file, zone #    2
455.    This grid contains IBLANK data which will be read.
456.    -----

457.    Constructing subgrid #    1 of    3 subgrids in zone    2
458.    Total count: subgrid #    4 of   16 subgrids
459.    Grid cells overlap 1 cell widths on grid boundaries.
460.    IBLANK data available and will be written to
461.    this subgrid file.
462.    nz =    2 jcube =    1 kcube =    1 lcube =    1
463.    jp3d_min =    1  jp3d_max =    3
464.    kp3d_min =    1  kp3d_max =   85
465.    lp3d_min =    1  lp3d_max =   20

466.    Constructing subgrid #    2 of    3 subgrids in zone    2
467.    Total count: subgrid #    5 of   16 subgrids
468.    Grid cells overlap 1 cell widths on grid boundaries.
469.    IBLANK data available and will be written to
470.    this subgrid file.
471.    nz =    2 jcube =    1 kcube =    2 lcube =    1
472.    jp3d_min =    1  jp3d_max =    3
473.    kp3d_min =   84  kp3d_max =  167
474.    lp3d_min =    1  lp3d_max =   20

```

475. Constructing subgrid # 3 of 3 subgrids in zone 2  
476. Total count: subgrid # 6 of 16 subgrids  
477. Grid cells overlap 1 cell widths on grid boundaries.  
478. IBLANK data available and will be written to  
479. this subgrid file.  
480. nz = 2 jcube = 1 kcube = 3 lcube = 1  
481. jp3d\_min = 1 jp3d\_max = 3  
482. kp3d\_min = 166 kp3d\_max = 249  
483. lp3d\_min = 1 lp3d\_max = 20

484. -----  
485. Reading original grid file, zone # 3  
486. This grid contains IBLANK data which will be read.  
487. -----

488. Constructing subgrid # 1 of 3 subgrids in zone 3  
489. Total count: subgrid # 7 of 16 subgrids  
490. Grid cells overlap 1 cell widths on grid boundaries.  
491. IBLANK data available and will be written to  
492. this subgrid file.  
493. nz = 3 jcube = 1 kcube = 1 lcube = 1  
494. jp3d\_min = 1 jp3d\_max = 3  
495. kp3d\_min = 1 kp3d\_max = 82  
496. lp3d\_min = 1 lp3d\_max = 20

497. Constructing subgrid # 2 of 3 subgrids in zone 3  
498. Total count: subgrid # 8 of 16 subgrids  
499. Grid cells overlap 1 cell widths on grid boundaries.  
500. IBLANK data available and will be written to  
501. this subgrid file.  
502. nz = 3 jcube = 1 kcube = 2 lcube = 1  
503. jp3d\_min = 1 jp3d\_max = 3  
504. kp3d\_min = 81 kp3d\_max = 161  
505. lp3d\_min = 1 lp3d\_max = 20

506. Constructing subgrid # 3 of 3 subgrids in zone 3  
507. Total count: subgrid # 9 of 16 subgrids  
508. Grid cells overlap 1 cell widths on grid boundaries.  
509. IBLANK data available and will be written to  
510. this subgrid file.  
511. nz = 3 jcube = 1 kcube = 3 lcube = 1  
512. jp3d\_min = 1 jp3d\_max = 3  
513. kp3d\_min = 160 kp3d\_max = 240  
514. lp3d\_min = 1 lp3d\_max = 20

515. -----  
516. Reading original grid file, zone # 4  
517. This grid contains IBLANK data which will be read.  
518. -----

519. Constructing subgrid # 1 of 2 subgrids in zone 4  
520. Total count: subgrid # 10 of 16 subgrids  
521. Grid cells overlap 1 cell widths on grid boundaries.  
522. IBLANK data available and will be written to  
523. this subgrid file.  
524. nz = 4 jcube = 1 kcube = 1 lcube = 1  
525. jp3d\_min = 1 jp3d\_max = 3  
526. kp3d\_min = 1 kp3d\_max = 69  
527. lp3d\_min = 1 lp3d\_max = 20

528. Constructing subgrid # 2 of 2 subgrids in zone 4  
529. Total count: subgrid # 11 of 16 subgrids  
530. Grid cells overlap 1 cell widths on grid boundaries.  
531. IBLANK data available and will be written to  
532. this subgrid file.  
533. nz = 4 jcube = 1 kcube = 2 lcube = 1  
534. jp3d\_min = 1 jp3d\_max = 3  
535. kp3d\_min = 68 kp3d\_max = 135  
536. lp3d\_min = 1 lp3d\_max = 20

537. -----  
538. Reading original grid file, zone # 5  
539. This grid contains IBLANK data which will be read.  
540. -----

```

541. Constructing subgrid # 1 of 3 subgrids in zone 5
542. Total count: subgrid # 12 of 16 subgrids
543. Grid cells overlap 1 cell widths on grid boundaries.
544. IBLANK data available and will be written to
545. this subgrid file.
546. nz = 5 jcube = 1 kcube = 1 lcube = 1
547. jp3d_min = 1 jp3d_max = 3
548. kp3d_min = 1 kp3d_max = 80
549. lp3d_min = 1 lp3d_max = 20

550. Constructing subgrid # 2 of 3 subgrids in zone 5
551. Total count: subgrid # 13 of 16 subgrids
552. Grid cells overlap 1 cell widths on grid boundaries.
553. IBLANK data available and will be written to
554. this subgrid file.
555. nz = 5 jcube = 1 kcube = 2 lcube = 1
556. jp3d_min = 1 jp3d_max = 3
557. kp3d_min = 79 kp3d_max = 157
558. lp3d_min = 1 lp3d_max = 20

559. Constructing subgrid # 3 of 3 subgrids in zone 5
560. Total count: subgrid # 14 of 16 subgrids
561. Grid cells overlap 1 cell widths on grid boundaries.
562. IBLANK data available and will be written to
563. this subgrid file.
564. nz = 5 jcube = 1 kcube = 3 lcube = 1
565. jp3d_min = 1 jp3d_max = 3
566. kp3d_min = 156 kp3d_max = 234
567. lp3d_min = 1 lp3d_max = 20

568. -----
569. Reading original grid file, zone # 6
570. This grid contains IBLANK data which will be read.
571. -----

572. Constructing subgrid # 1 of 2 subgrids in zone 6
573. Total count: subgrid # 15 of 16 subgrids
574. Grid cells overlap 1 cell widths on grid boundaries.
575. IBLANK data available and will be written to
576. this subgrid file.
577. nz = 6 jcube = 1 kcube = 1 lcube = 1
578. jp3d_min = 1 jp3d_max = 3
579. kp3d_min = 1 kp3d_max = 68
580. lp3d_min = 1 lp3d_max = 20

581. Constructing subgrid # 2 of 2 subgrids in zone 6
582. Total count: subgrid # 16 of 16 subgrids
583. Grid cells overlap 1 cell widths on grid boundaries.
584. IBLANK data available and will be written to
585. this subgrid file.
586. nz = 6 jcube = 1 kcube = 2 lcube = 1
587. jp3d_min = 1 jp3d_max = 3
588. kp3d_min = 67 kp3d_max = 133
589. lp3d_min = 1 lp3d_max = 20

590. -----
591. Screen output directed to file: BREAKUP.OUT
592. Subgrids output to file: GRID_DIVIDED.G
593. in PLOT3D format.
594. -----

595. >> Finished <<

596. Program BREAKUP finished.

```

EXTERNAL DISTRIBUTION:

5	Micro Craft Attn: J. A. Benek (1) N. Suhs (4) 207 Big Springs Avenue P.O. Box 370 Tullahoma, TN 37388-0370	1	Dr. Dave Bader Battelle Washington Operations 901 D Street SW, Suite 900 Washington, DC 20024-2115
1	Dr. C. Wayne Mastin Nichols Research 2524 South I-20 Frontage Rd. Suite A P.O. Box 820186 Vicksburg, MS 39182	1	Dr. Pieter Buning Configuration Aerodynamics Branch MS 499 NASA Langley Research Center Hampton, VA 23681-0001
1	M. Remotigue NSF/ERC at MSU MS 962 ERC 203 P.O. Box 9627 Mississippi State, MS 39762	1	Dr. F. C. Dougherty Supercomputer Computations Research Institute Florida State University Tallahassee, FL 32306-4052
1	Dr. Ray Gomez Advanced Programs Office Mail Code EG3 NASA Johnson Space Center Houston, TX 77058	1	Dr. Stuart Rogers Mail Stop 227-2 NASA Ames Research Center Moffett Field, CA 94035-1000
1	Prof. David Whitfield Engineering Research Center Mississippi State University P.O. Box 9627 Mississippi State, MS 39762	3	Charlie Nietubicz, Director Major Shared Resource Center US Army Research Laboratory Aberdeen Proving Ground, MD 21005-5067
3	Dr. Bob Meakin NASA Ames Research Center MS 258-1 Moffett Field, CA 94035-1000	1	Dennis Jespersen MS T27B-1 NASA Ames Research Center Moffett Field, CA 94035-1000
5	Lockheed Martin Skunkworks Attn: G. Shrewsbury (4) J. Vadyak (1) Computational Aerodynamics 1011 Lockheed Way Palmdale, CA 93599-2523	2	CFD Research Corporation Attn: Sami D. Habchi 215 Wynn Drive Huntsville, AL 35805
1	Mr. D. Howlett Engineering Chief Aerodynamics & Computational Fluid Dynamics Lockheed Martin Tactical Aircraft Systems PO Box 748 Mail Zone 9333 Ft. Worth, TX 76101	2	Dr. Michelle Hribar Dr. Jerry C. Yan MS T27A-2 NASA Ames Research Center Moffett Field, CA 94035-1000

INTERNAL DISTRIBUTION:

1	MS	0318	G. S. Davison, 9215
1		0321	W. J. Camp, 9200
1		0439	D. R. Martinez, 9234
1		0441	P. Knupp, 9226
1		0441	R. W. Leland, 9226
1		0819	J. Peery, 9231
1		0820	P. Yarrington, 9232
1		0825	B. Hassan, 9115
1		0825	J. Payne, 9115
1		0825	F. Blottner, 9115
1		0825	W. Rutledge, 9115
1		1109	A. L. Hale, 9224
1		1110	D. Greenburg, 9223
1		1110	D. E. Womble, 9222
1		1111	S. S. Dosanjh, 9221
5		1111	D. W. Barnette, 9221
1		1111	G. Heffelfinger, 9225
1		1166	J. D. Kotulski, 9352
1		1166	D. J. Riley, 9352
1		1166	D. C. Turner, 9352
1		0161	Patent & Licensing Office, 11500
1		9018	Central Technical Files, 8940-2
2		0899	Technical Library, 4619
2		0619	Review & Approval Desk, 12690 For DOE/OSTI