# COOPERATIVE SOLUTIONS COUPLING A GEOMETRY ENGINE AND ADAPTIVE SOLVER CODES

Thomas P. Dickens
Aerodynamics Configuration Computing
The Boeing Company
Seattle, WA

## ABSTRACT

Follow-on work has progressed in using Aero Grid and Paneling System (AGPS[1,2]), a geometry and visualization system, as a dynamic real time geometry monitor, manipulator, and interrogator for other codes, as first published in 1992[3]. In particular, AGPS has been successfully coupled with adaptive flow solvers which iterate, refining the grid in areas of interest, and continuing on to a solution. With the coupling to the geometry engine, the new grids represent the actual geometry much more accurately since they are derived directly from the geometry and do not use refits to the first-cut grids. Additional work has been done with design runs where the geometric shape is modified to achieve a desired result. Various constraints are used to point the solution in a "reasonable" direction which also more closely satisfies the desired results.

Concepts and techniques are presented, as well as examples of sample case studies. Issues such as distributed operation of the cooperative codes versus running all codes locally and pre-calculation for performance are discussed. Future directions are considered which will build on these techniques in light of changing computer environments.

## INTRODUCTION

Current computing technology enables increasingly complex geometric configurations to be analyzed and allows analysis at a much finer level of detail. Flow solvers must work with the geometric definitions more closely to accommodate the level of accuracy desired. Rather than building more sophisticated geometry capabilities into the solvers, which is a duplication of the effort that went into the geometry code, a cooperative environment can be created which facilitates multiple codes working together. This environment will use the specialized capabilities and knowledge of each system, rather than investing the time and money to replicate capabilities in multiple codes plus the incurred problem of diverging capabilities once they are duplicated. The application of the code cooperation is needed in the geometry/solver combination due to the complexity and specialization of each of the two fields. This coupling of codes is especially beneficial in adaptive solvers, and when using solvers in a geometry-design mode.

In an aerodynamics design mode, a pressure solution is first generated for a geometric configuration. This pressure solution is then examined and modified by an engineer to represent a desired condition using a method of their choice. An area of the geometric configuration is selected for modification and may have associated constraints imposed. The solver then systematically perturbs the geometry while tracking the effects on the resulting solution. A new geometric configuration is calculated based on the results of the perturbations. A solution is then run for this new geometric configuration, which is then compared with the desired solution. This process is iterated as required.

It should be pointed out that, in this scenario, the geometric configuration is paneled, and then the resulting paneling is provided to the solver. The solver then perturbs the paneling, which represents the geometric configuration, to achieve the desired solution. The assumption with the paneling is that it is sufficiently refined to represent the actual surfaces within a desired tolerance. In these design runs, when the paneling is modified to represent new geometry, the original tolerances used to

generate the paneling may now be greater than desired to accurately represent the geometry. The solver itself does not understand the original geometry, nor does it know the original criteria which went into the paneling decisions. I have seen design cases which generated fairly sharp grooves and bulges in surfaces, as seen in Figure 1. If the original paneling was just within a desired criteria (chord-height tolerance with the geometry for example) changes to the design region can easily generate panels which, if tested back against the original geometry, are no longer within tolerance. This may introduce unwanted artifacts into the solution which degrade the usefulness of the achieved solution. Once the design run has achieved a target geometry which satisfies the desired pressure, the new geometric shape must be generated in the geometry tool (not a trivial task), a new paneling created, and a solution run on the new geometry to test the modified geometry.

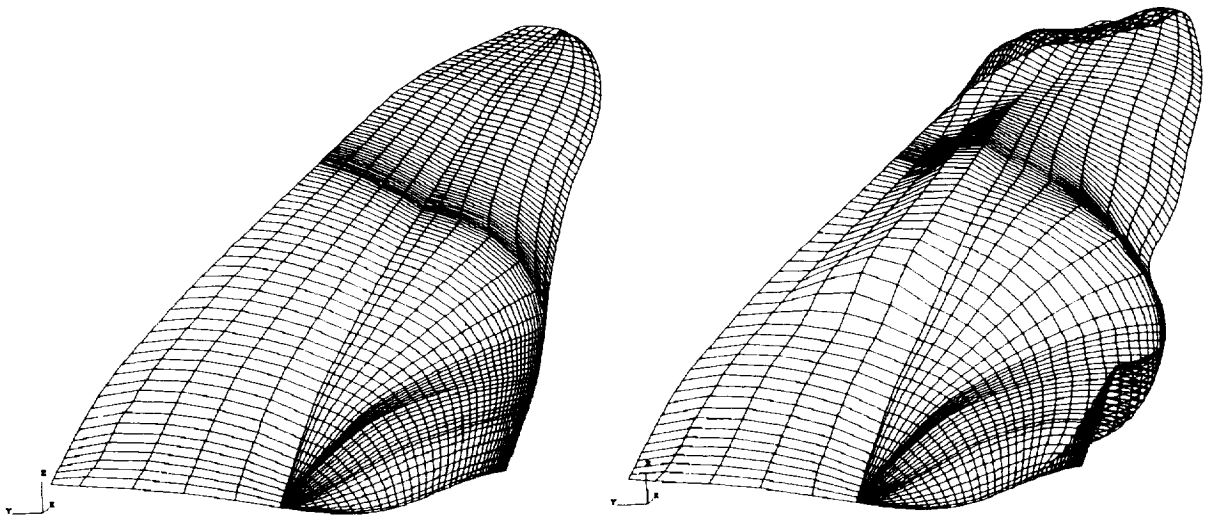

Figure 1. Before (left) and After (right) Panels of a Strut.

Another concern is the use of constraints within the solver. Localized constraints are used which can constrain the resulting grid to fit within a curvature tolerance, for example, or to impose a constraint to only allow movement in the positive surface-normal direction. Localized constraints are the first step in the process, but they do not have the necessary influence on the global properties of the configuration. Without more global constraints, local constraints can generate oscillations or other undesired artifacts in the resulting paneling which are out of the scope of the local constraints to "see".

The last step in the process is to create a new geometry loft which represents the modified panel data: the resulting output from the design-mode solver. The input panels were originally generated by discretizing the surface lofts within the geometry tool. The surface lofts are typically generated from a series of airfoils in the case of wing-like surfaces, or station curves for body-like surfaces. It should be noted that where two surfaces intersect, such as a wing-body intersection, the surface loft of the wing typically extends into the body. The extracted paneling is only for the exposed wing surface, the subrange area on the wing which is outboard of the wing-body intersection. The modified panel networks which are returned by the solver are used as templates to generate a new loft.

Several challenges occur here. There may be geometric shapes such as bumps and grooves in the new panels which are located between the original airfoils used for the original loft. Additional airfoils may be required for the new loft to sufficiently capture these shapes. But there are problems with using too many airfoils; high frequency noise can be introduced in the loft. Another problem is re-creating the parts of the loft that were not paneled, such as the part of the wing which is inside the body. The original loft data may be used, but if there are changes required in this region, the original data may not be useful. In this process you also wonder about all of the ripples in the new panels: are they necessary to achieve the desired solution, or are they artifacts of the process and not absolutely necessary? Once a new loft is generated, it is then paneled and processed again by the

solver with the hopes of matching the previously obtained results achieved from the design run.

## TERMINOLOGY OVERVIEW

To facilitate communication with a common understanding, some foundation definitions and concepts are outlined:

Geometric Definition.–Geometry is defined by two criteria: data, and functions which utilize the data. For example, a 3D surface is represented using a 2D parameter space. A function for evaluating the surface will return surface properties for a given parametric pair: D = F(S, T), where the returned data, D, is the data at the given S and T parametric location on the surface, and can consist of the 3D spatial coordinates, derivative information, surface normal, and other associated information. Higher dimensionality can be used to define geometry which would also be accessed by the given function. A surface can be constructed through an array of 3D data points to yield a set of data. For example, if the geometric form is a series of bicubic patches in 3D space, each patch will have 48 pieces of data to define the physical points at the four patch corners, plus the parametric derivatives with respect to S, T, and the ST cross derivatives. On top of this is the organization of the multiple surface patches to the parent surface. The surface evaluation routines are synchronized with the surface construction routines within a geometry tool. Without visibility of the methods and equations involved within the evaluation function, other codes can only approximate the interpretation of the geometry defined by the data. Additional complexity is added when the geometry tool builds upon the geometric definitions with techniques such as subrange and trimmed surfaces as well as procedural entities.

The key point here is that the data alone is not enough to accurately represent geometry. The defined evaluation process, or geometry engine, to work with the data is also required. Two different evaluation processes operating on the same data may not produce the same results. Thus, even if a solver has the capability to work with geometric data rather than just paneling representation of that data, unless the solver uses the same exact code as the geometry definition tool, the resulting geometry as known to the solver will be different than the geometry as defined in the geometry tool.

The Paneling Burden.–In the current computational fluid dynamics (CFD) process, the following tasks are performed: creating geometry, paneling the geometry, running the CFD code, and analyzing the solution. The particular CFD code used here is a full potential code that uses a locally refined rectangular grid which is generated internally by the code. In terms of flow time, the paneling process continues to be a bottleneck in the overall CFD process. Engineers currently can spend days paneling a typical wing-body-strut-nacelle configuration, and the resulting quality of the CFD solution is directly affected by the quality of the paneling. The engineer is burdened with the task of ensuring that the paneling is detailed enough for the problems being addressed to achieve meaningful results, yet not too dense to break the solver code or take too much time to run. Common techniques are used to help automate the paneling process. Such techniques are based on chord-height tolerance of the flat panel to the surface or other methods. The paneling process is not a bottleneck if the task can be automated, such as by using AGPS command files written to handle the topology of the configuration.[4]

Another concern is in iterative adaptive solutions, where emerging details in the solution will concentrate volume grids in areas of interest, such as shocks. The volume grids can generally be easily refined to detail these areas of interest. However, where the volume grid meets with the paneling, the flow code typically does a simple interpolation of the paneling to determine the panel/field intersection. It is at this panel/field intersection where additional knowledge of the actual geometry is needed. If the paneling in this region is not fine enough to accurately represent the geometry to achieve a reasonable solution, the task must be redone with a modified panel. This requires engineers to study solutions, to refine the paneling, and then rerun the codes. This is costly both in time and money. Of even greater risk is that the effect of the nonsufficient paneling is not realized and that an inaccurate solution is believed to be more accurate than it is. This is because the CFD code deals only with discretized geometry (paneling) and not with the original surface lofts. Again, the burden is placed on the engineers to understand the limitations of their paneling and to interpret the given solution with the appropriate level of accurateness. Of course a given paneling

may be intended to show general tendencies and is not intended to be used for finer details.

AGPS Overview.–While developed and used primarily for the preliminary design of aircraft, AGPS is used throughout The Boeing Company (Boeing) for a variety of geometric tasks. This Boeing proprietary program is a surface geometry system in which virtually any shape can be modeled. The underlying mathematics include cubic and quintic polynomials and rational B-splines for representing curves, surfaces, and solids. The interface consists of a structured programming language with over 160 geometry-related commands, along with a mouse-and-menu-driven interface. Higher-level command files can be constructed and used as a macro capability to do complex or repetitive tasks very simply. Collections of command files are used as high-level packages to allow users to accomplish complex tasks by following an interactive menu-driven session. Hundreds of existing command files are included in the AGPS release, which runs on a variety of machines including VAX, SGI, HP/Apollo, IBM, and Cray. AGPS has been coded to be machine and architecture independent, and utilizes a dynamically allocated object data structure.

TRANAIR Overview.[5]–A full-potential, solution-adaptive, rectangular grid code for predicting subsonic, transonic, and supersonic flows about arbitrary configurations, TRANAIR is used to analyze complex geometric configurations in transonic flow. A locally refinable rectangular grid is automatically constructed and is superimposed on the boundary geometry as described by networks of panels. Surface-fitted grid generation is not required. The nonlinear discrete system is solved using a preconditioned Krylov subspace method embedded in an inexact Newton method. The solution is obtained on a sequence of successively refined grids.

## GEOMETRY USE IN SOLVERS

There is an increasing trend to use solvers in a geometric design mode, in which the solver will be modifying the geometry definitions. To successfully accomplish this, greater attention must be placed on the geometry operations as driven by the solvers. There are two main directions to accomplish this: additional geometric capabilities can be coded into solvers, or the geometry tool and the solver can work together.

The geometric sophistication within solvers is limited. This is understandable and even expected; their domain of expertise is flow solving, and one does not expect to have solvers as sophisticated as dedicated geometry tools in the domain of geometry and geometric definitions and operations. Incorporating geometric sophistication within solvers makes the solvers much more complicated, and duplicates capabilities among various codes. Even if this route is taken, the geometric definitions will not be exact unless the same exact code used in the geometric tool is also used by the flow solver. The natural evolution for this path is to evolve the solver to also be your geometry tool, or to add solver capabilities to your geometry tool. Each of these fields require specialized expertise and generate extremely complex systems as they currently are. If a solver and geometry tool are merged into a single code, the resulting code could be too complex to manage.

A logical alternative approach is to dynamically couple the solver code with the geometry code, each working in a cooperative manner on the part of the problem which is its specialty. This approach has advantages and disadvantages. The primary advantage is the ability to refer back to the original geometric definitions. This is used for panel refinements as well as for moving the geometry and repaneling the changed area. By definition, in this process the paneling will define the geometry to the specified accuracy. Geometry changes will be transformed into new paneling which maintains the desired accuracy in its representation. With the geometry tool used in the design process, changes to the paneling can be applied to the loft process instead, which will then produce a modified loft. New paneling is then generated from this new loft. Once the design process achieves an acceptable solution, the loft which generated the paneling for that solution is already there. The problem of creating a new loft to represent the results of the design process is eliminated.

The disadvantages to this approach are the requirement of enabling the codes to cooperate together, as well as the performance overhead of the communications between the codes. Once the direction to enable codes to cooperate is taken, relatively minor modifications can be done to the

codes involved to accomplish this task. This leaves the performance penalty in the communications between the codes as the major disadvantage.

Compared with the alternatives of either working in the current paradigm or incorporating the modifications necessary to put sophisticated geometric capabilities into a solver (or the reverse), I contend that the advantages of coupling these codes strongly outweigh the disadvantages.

## COUPLING MULTIPLE CODES

Within Boeing, work has been done in this direction. I have added the capabilities to AGPS to allow various cooperation methods between codes. This cooperation can be set up in two different schemes, based on which programming unit (AGPS or another code) in the system is in control.

External control, or what we call the *master* type of connection, allows another program to connect to AGPS and directly interface with the AGPS command line input stream. The external process also will receive all of the output from AGPS (Figure 2). In effect, the external master program appears to AGPS to be a user interacting with the normal input and output capabilities of AGPS. This capitalizes on a very successful feature of AGPS: the command-line interface. The AGPS development team has recognized the importance of offering almost all of the capabilities within AGPS in a command-line interface as well as a graphical user interface (GUI) point-and-click interface. This feature allows repetitive tasks to be easily automated and repeated with little or no user interaction required. Routines which initiate and terminate the controlled AGPS connection, as well as send and receive data between AGPS and the master program, are provided by the AGPS development team to allow this capability to be easily incorporated into the I/O system of a master program. This feature allows an external program to easily assume the role of the AGPS user and to have AGPS accomplish geometric tasks for the program.

There is a second level of master program access into AGPS. The external program can directly invoke a set of AGPS internal subroutines which handle the geometry creation, modifications, and interrogation within AGPS. The direct routine access capabilities are generally not used by external programs; the command-line interface is the most widely used method to control AGPS via an external program.

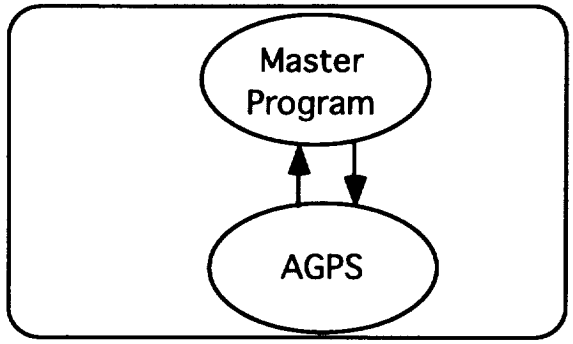


Figure 2. Master Program Using AGPS Capabilities.

The second scheme is to have AGPS in control. Up to nine external processes, referred to as *slave* processes, can be dynamically invoked and connected to AGPS (Figure 3). A slave process is spawned from AGPS, with AGPS remapping the standard-in and standard-out streams of the slave process into pipes which are held by AGPS. This way the slave process does not need to be modified in any way to allow it to be used as an AGPS slave process. This was a very important concept for the general usefulness of the slave capability. If two communicating programs need to jointly agree on an interface specification and protocol and need to be maintained with the knowledge of the other, the interdependence between the codes may be unworkable. In addition, the coupling of a different pair of codes will require modifications to both of them to allow them to communicate. With the approach of mapping the standard-in and standard-out streams of the invoked process, the invoked

process does not require any modifications. Any process or program available on the system can be utilized as a slave process. This allows AGPS to have at its command an endless supply of dynamic capabilities. This is used in production to provide added capability to AGPS, and to prototype new AGPS capabilities without requiring modifications to AGPS.

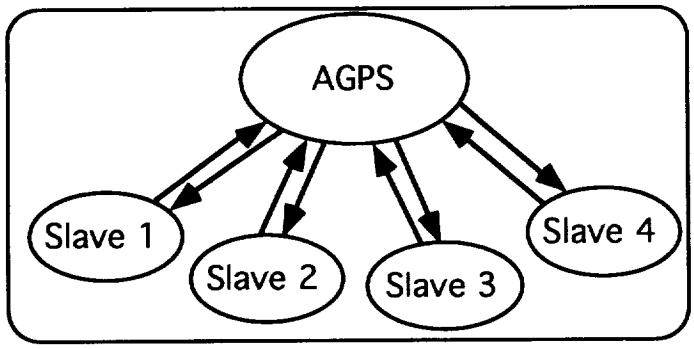


Figure 3. AGPS Using Multiple Slave Programs.

Tradeoffs.–With the techniques of coupling two or more codes to cooperatively accomplish a task, there are costs and compromises involved. On the positive side, the savings in development time to incorporate major new capabilities are significantly reduced from months or years into a few hours or days to access the capabilities through existing external codes. This provides the new capabilities almost immediately when needed. Specialized expertise is not required to develop the capabilities from scratch, and the capabilities are known to be mature and robust. On the negative side of code coupling are performance issues. With two or more processes communicating, there will be an overhead involved with physically running the programs. More important is the limited bandwidth for communication between the processes and the sequential nature of tasks. These issues can be addressed and minimized while still maintaining the general-purpose nature of the solution.

However, attaining the same performance as compared to a dedicated program is not achievable. My feeling is that if the overall solution is no more than an order of magnitude slower, the performance penalty is acceptable, at least for initial proof of concept purposes. The bottleneck of the communication overhead can be minimized through the use of files to communicate large quantities of information, using process-to-process communication primarily for control. Another technique is to partition the task in ways that minimize the communication required and allow for larger amounts of work to be accomplished for a given set of data. This can be combined with the concern of sequential work by front-loading the known work to be done with an external system to allow the external system to have as much of the work completed when the master system requires the results. As demonstrated below, external tasks can be front-loaded and also distributed among multiple machines to achieve more responsive turnaround.

## CASE STUDY: TRANAIR AND AGPS

As a case study of these techniques, we will consider the use of TRANAIR in a design mode which uses AGPS as a geometry engine.

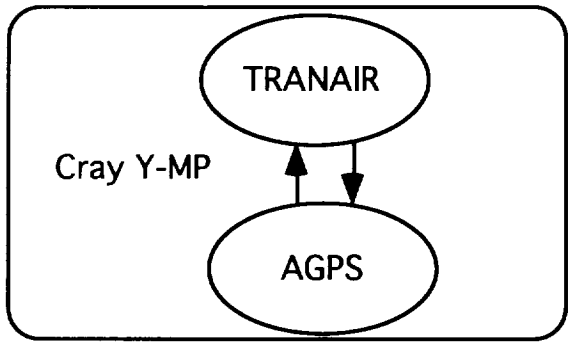


Figure 4. Coupling of TRANAIR and AGPS on the Cray.

The first implementation of the TRANAIR/AGPS coupling used both codes running simultaneously on the Cray Y-MP (Figure 4). It should be noted that TRANAIR has been optimized for the Cray architecture, while AGPS has not. Add to this the interpreted command-line nature of AGPS which results in a very generalized geometry system, but a system which is much slower than dedicated code to accomplish specific tasks. Code was added to TRANAIR to launch AGPS, and to communicate the desired geometry tasks to AGPS. The algorithms involved to solve the geometry tasks were implemented in AGPS command files which allowed easy development and modifications to the algorithms without requiring subsequent changes to either of the two systems. TRANAIR requested AGPS to perform a geometry task and then waited for AGPS to finish the task. AGPS communicated the result to TRANAIR via a data file, which TRANAIR read. For a sample design case using 40 design points, 80 solutions were required of AGPS for each iteration. Needless to say, this was slower than implementing similar capabilities directly into TRANAIR. Each iteration using AGPS took about a minute to complete, and then TRANAIR took a few seconds to digest the information. A similar implementation coded solely into TRANAIR took a few seconds per iteration. These results must be weighed with the fact that the AGPS solution was configured to run within a couple of days while a TRANAIR version modified with similar capabilities took weeks (flow time) to be available, and the capabilities were job-specific and hard-coded into the code.

Much was learned from these experiments. A small change in TRANAIR was made which launched the 80 AGPS jobs as soon as TRANAIR knew what those jobs consisted of--which was about 10 minutes before TRANAIR required the results. This gave AGPS some lead time to get a head start on the task. It was also noted that the initial conditions were known before the run was even started, so the first iteration results could be pre-calculated by AGPS before the TRANAIR run was initiated. During the process we were also able to gather metrics on the running of TRANAIR, which allowed bottlenecks within the TRANAIR design-mode code to be discovered and optimized.

The next logical step in the experiment was to distribute the AGPS tasks to a local workstation (Figure 5). Since AGPS is not optimized for the Cray architecture, wall-time performance on workstation class machines was comparable to the shared Cray performance. Using a handful of UNIX scripts and files for communications between the two machines, running TRANAIR on the Cray and the AGPS geometry jobs on a workstation gave comparable results to running both codes on the Cray. This has the added benefit of off loading the geometry task from the Cray which allowed our Cray resources to be focused on the solver task. Even with the 10 minute lead time, AGPS still could not keep up with TRANAIR's need for results when it required them.

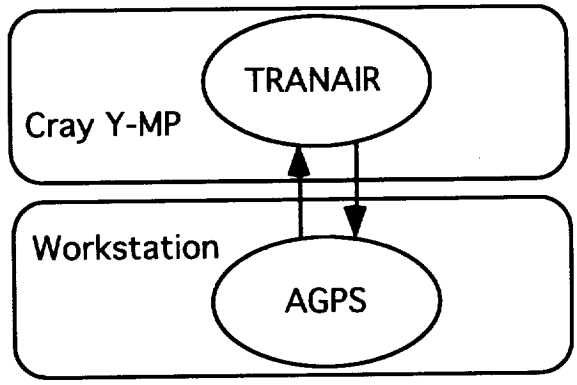


Figure 5. TRANAIR and AGPS Coupled Across Different Machines.

We then looked at distributing the geometry task across multiple workstations. This was initially done using UNIX scripts to launch remote AGPS sessions on predetermined machines and to assign a predetermined set of jobs to each machine (Figure 6). A central machine was used to collect the results and communicate them to the Cray. When this scheme worked, it worked well. We were able to feed the results to the Cray fast enough to keep up with its demand; however, this scheme is quite fragile in practice. With the list of machines predetermined, and the jobs on these machines predetermined, if one machine was either unavailable or heavily loaded during the run, the job would fail due to an incomplete set of results. Some system-level glitches were discovered, such as the inability to spawn remote jobs on the Cray with a large number of files opened. We finally tracked this glitch into the system kernel on the Cray as a problem with the rsh command.

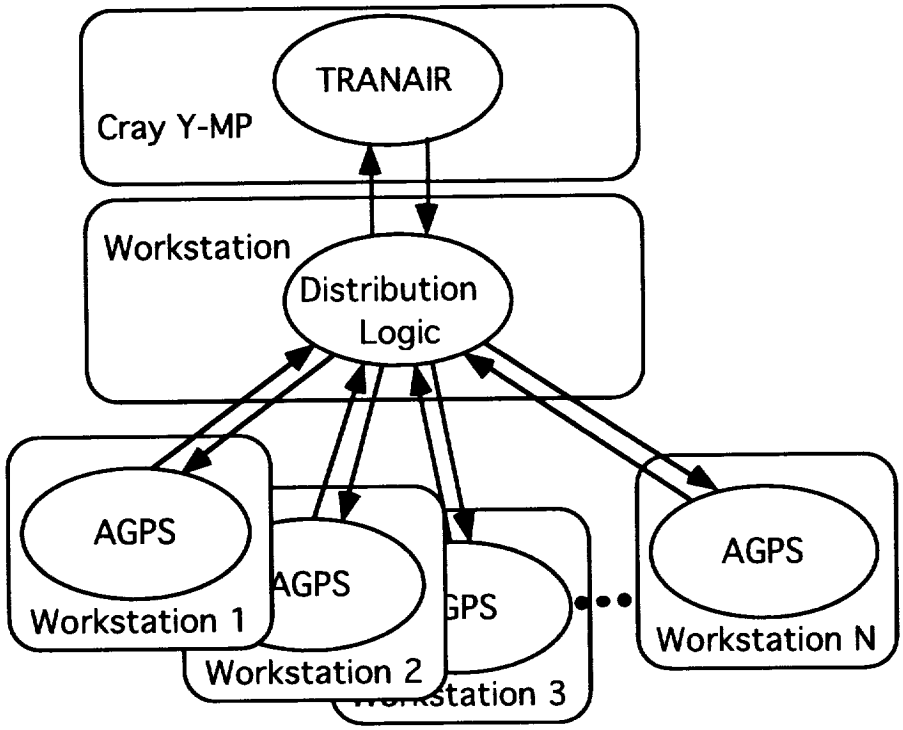


Figure 6. TRANAIR with Multiple AGPS Coupled Across Many Machines.

We are currently working to refine these techniques and to offer them to the Boeing user community in an easy-to-use manner.

## CONCERNS AND FUTURE WORK

During this effort we discovered various roadblocks and have identified areas needing enhancements.

Receiving Complete Files.–Three problems may occur in the transmission of files between the various machines. One problem is the corruption of a file. This is rather rare due to the robustness and error checking built into ftp and rcp. A second problem is the non-arrival of a file. This does happen and can be caused by network problems, file-space problems, or code failure on the sending machine. A third problem occurs rather commonly; the use of a file before the entire file is received. There is no mechanism built into UNIX to make a file unavailable until it is complete. Various techniques can be used to address this, such as including the checksum of the file as a field in the name of the file. The receiving program will not open or access the file until the checksum generated from the file matches the checksum in the filename. This has the added benefit of also checking for corrupt files. Another method is for the sending machine to send the file with a temporary filename, then, after transmission, to remotely change the name of the file. This should work in theory, but subtle timing through the network can cause failure. The transmission can be completed and then the file name changed, while actually the file is not completely available on the remote system.

Working with Dynamic Machine/Network Loading.–At this time, work is being done on a C program which will dynamically determine the health of a given set of machines on the network and issue geometry tasks to be solved with AGPS jobs running on the subset of available machines. These tasks will be monitored for completion and also for performance. Remaining jobs will be allocated in a logical manner to ensure completion of the entire set of tasks as quickly as possible as well as the ordering of the results received. Tasks may be reissued on top performers if the assigned machine is not responding in a timely manner, or if the resulting files are not complete. This approach will offer many improvements. Most failures in the system can be recovered from during a run, including a node being unavailable, a node being too heavily loaded with other tasks, network failures, and file corruption. Problems can still occur if the central machine coordinating the distributed effort goes down, loses its network connection, or is too heavily loaded. Preliminary testing of this capability looks very promising.

## SUMMARY

Benefits of using AGPS for geometry tasks.–It has been demonstrated that there are a variety of benefits in coupling two or more specialized programs together in a cooperative manner. These benefits must be weighed with the costs incurred to determine the appropriateness of this technique for a given environment and problem. The costs include both development costs and run-time costs. There will be a development effort involved to set up the programs for cooperative communication. Doing the development changes in a general-purpose manner, and implementing the main capability in a key central program which does not dictate changes in other programs, is highly favorable. The run-time costs include the overhead of running multiple codes and running codes not optimized for a particular task. This results in both higher CPU usage costs as well as longer wall-time turnaround. These costs can be minimized by partitioning the problem. Benefits include the rapid availability of new capabilities, use of capabilities without requiring extensive code development, gaining access to specialized code without duplicating or reproducing the code, and immediate access to robust, mature code.

Easy-to-change tasks (command-file driven and dynamically accessed capabilities).–By using the AGPS command language to accomplish the geometry tasks, algorithmic changes to the command files can be incorporated without requiring changes to any compiled source code. This allows very rapid turnaround of modifications to the geometric algorithms. In addition to production work, this facilitates rapid prototyping of new capabilities without requiring the system to be modified and rebuilt. The expertise needed to work in this environment does not require an expert with the internals of either code.

Work is done on the actual geometry, in the defining system of the geometry.–Refinements made at the request of the solver can be made to the actual geometry, rather than using extracted panels and

approximating the defining geometry. As geometric changes are made, effects to the surfaces can be realized and used. Dynamic refinements to the paneling is possible to accommodate areas of interest as the solution evolves.

AGPS has the tools to work with global constraints/conditions.–Using the capabilities in a mature geometry system provides a much richer set of offerings than is available as add-on capabilities in a solver. These offerings are also proven to be reliable due to widespread use for other applications over time, while new code added into solvers will not be mature or widely tested. Many years of work have gone into the geometry system, which results in a system which is fine tuned to work with geometry, both on a local scale and in a global scale. This global understanding of the geometry, and the ability to work with the geometry in this manner, allows better constraints to be imposed on the solution which will lead to more usable solutions.

Changing loft definitions in the design mode.–By using AGPS to dynamically modify the loft definition during a design run, the problem of creating a new loft based on a set of modified panel networks is eliminated. The cost to run a solution for the new loft is also eliminated. The result of the design run is a new loft rather than modified panel networks.

Generality–The techniques presented here are appropriate to be applied in a general manner to numerous other applications and should not be limited to CFD solvers and geometry codes. Candidate applications are those where two or more specialized codes working together can yield a cooperative benefit.

## ACKNOWLEDGMENTS

I would like to thank Arvel Gentry for his shared vision and his long hours using and evolving these techniques. I also appreciate the cooperation and knowledge from the TRANAIR development group and the other members of the AGPS development group.

## REFERENCES

1. D. K. Snepp and R. C. Pomeroy, "A Geometry System for Aerodynamic Design," AIAA/AHS/ASEE Aircraft Design, Systems, and Operations Meeting, AIAA-87-2902, September 14-16, 1987.

2. W. K. Capron and K. L Smit, "Advanced Aerodynamic Application of an Interactive Geometry and Visualization System," 29th Aerospace Sciences Meeting, AIAA-91-0800, January 7-10, 1991.

3. T. P. Dickens, "Technique for Using a Geometry and Visualization System to Monitor and Manipulate Information in Other Codes," Software Systems for Surface Modeling and Grid Generation, NASA Conference Publication 3143, April 28-30, 1992.

4. A. E. Gentry, "Requirements for a Geometry Programming language for CFD Applications, Software Systems for Surface Modeling and Grid Generation," NASA Conference Publication 3143, April 28-30, 1992.

5. F. T. Johnson et al., "TRANAIR: A Full-Potential, Solution-Adaptive, Rectangular Grid Code for Predicting Subsonic, Transonic, and Supersonic Flows About Arbitrary Configurations," Theory Document, NASA Contractor Report 4348, December 1992.