APPLICATION OF THE pV3 CO-PROCESSING VISUALIZATION ENVIRONMENT TO 3-D UNSTRUCTURED MESH CALCULATIONS ON THE IBM SP2 PARALLEL COMPUTER

Robert Haimes
Principal Research Engineer
Department of Aeronautics and Astronautics
Massachusetts Institute of Technology
Cambridge, MA 02139
haimes@orville.mit.edu
(617) 253-7518
and
Timothy Barth
Advanced Computational Methods Branch
NASA Ames Research Center
Moffett Field, CA 94035
barth@nas.nasa.gov
(415) 604-6740

Introduction

The **pV3** visualization system developed at MIT provides a series of graphics and visualization tools suited to co-processing computing environments using PVM message passing protocol. Efficiency and real time interactive response is achieved by data extraction so that a small fraction of the data present during the flow field computation is actually transferred to the graphics co-process. This requires that the solver be closely coupled to the data extraction portion of the visualization system. The result is an interactive CFD system that provides the investigator all of the traditional tools and probes found in most visualization post-processors.

The Flow Solver

The compressible fluid flow calculations are performed using a recently developed Euler and Navier-Stokes solver for 3-D unstructured meshes[1]. The solver uses a coupled one equation turbulence with the Navier-Stokes equations. The equations are discretized in space in an integral conservative form using a generalization of Godunov's upwind method for unstructured meshes. Linear spatial reconstruction of solution gradients is used to improve spatial accuracy. Rapid convergence to steady-state is obtained using an iterative scheme which combines the Generalized Minimum Residual method (GMRES) preconditioned with Incomplete Lower-Upper (ILU) factorization.

This solver has been fully parallelized for the SP2 computer using MPI message passing protocol. The volume mesh can be pre-partitioned by either coordinate bisection, Cuthill-McKee ordering, or spectral bisection. The result is N non-overlapping partitions. In any case, Cuthill-McKee node ordering is used for the resultant sub-domain to reduce cache memory misses. Residuals on partition boundaries are communicated for control volumes split by the interface. The GMRES

algorithm is fully parallelized but the ILU preconditioning is based only upon the local sub-domain.

The Visualization System

pV3[2] is a complete distributed CFD-style visualization package. It is new, but builds heavily from the technology developed for **Visual3**[3]. This re-working was necessary to support the visualization of transient problems without having to dump the entire volume of data to disk every iteration. **pV3** segregates the visualization system into two parts. The first part is the task that actually displays the visualization results onto the graphics workstation, the server. The second portion is a library of routines that allows the solver (solver portions or solvers) to communicate with the visualization server by providing windows to the data at critical times during the simulation. This client library separates all visualization based calculations from the flow solver, so that the programmer only needs to feed **pV3** the current simulation data.

PVM is used for all communication within the visualization system. For this application, the Ethernet interface into each node of the SP2 was used for the data interchange with the graphics workstation. PVM allows message passing in a heterogeneous cluster (SGI and the IBM SP2) and the Ethernet connections were used to get off the IBM SP2 and out to the location of the graphics workstation. PVM also provides a mechanism that allows the solver to be executed independently from the visualization. Groups are used to query which processes are part of the pool so that the server need not be given a list of active task IDs. This feature allows the visualization portion of this system to be run from any (SGI) graphics workstation that is part of the virtual machine. When the server terminates (the visualization session is over) the solution continues to run. The server may be re-started later and the state of the simulation examined, even from a different location.

pV3 has been designed to minimize network traffic. The client-side library extracts lower dimensional data required by the requested visualization tool from the volume of data in-place. This distilled data is transferred to the graphics workstation. To further reduce the communication burden posed by the visualization session, transient problems have been classified in the following manner:

Data Unsteady
: In this type of application the grid structure and position are fixed in time. The data defined at the nodes (both scalar and vector) changes with each time step. An example is when a boundary condition in the domain is changing.

Grid Unsteady
: These cases are ‘Data Unsteady’ plus the grid coordinates associated with each node are also allowed to move with each snapshot. An example of this is stator/rotor interaction in turbomachinery. The stator and rotor grids are separate, with the rotor grid sliding past the stator grid. In this case the stator portion is actually ‘Data Unsteady’ and the rotor grid moves in a radial manner.

Structure Unsteady
: If the number of nodes, number of cells or cell connectivity changes from iteration to iteration the case is ‘Structure Unsteady’. An example of this mode is store separation.

Each client may be assigned a different class. This tells the system if data needs to be recalculated and re-transmitted. For example, if a sub-domain is 'Data Unsteady' and a geometric cut is desired (such as a planar cut), only the field data at the nodes (scalar and/or vector) need to be re-transmitted to the server every iteration. The geometry (node locations and mesh data to support the surface) is communicated only when the cut is initiated. If, in the above example, the case is 'Grid Unsteady' all data needs to sent every iteration to server.

Coupling the Solver to pV3

In this specific application, the flow solver is time-marching towards a steady-state solution. To examine the solution as it converges, the clients are initialized indicating that the solver is 'Data Unsteady'. This is used as a debugging aid. The solution can be watched during unstable sequences, the failure modes examined, and coding or communication problems found.

When each solver task starts on a node in the IBM SP2, the MPI process ID is appended to the appropriate filenames in order that the grid and initial solution for the sub-domain may be read (over a distributed file system). This startup may take some time for large problems.

Like most CFD solvers this application spends most of it time in an iterative loop. The loop has the following components:

- Calculate strong boundary conditions
- Compute the right-hand-side
- Apply the weak boundary conditions
- Setup the matrices
- Perform the ILU preconditioned GMRES implicit scheme
- Communicate internal boundary modifications
- Update the field

The coupling is performed by adding two calls to the **pV3** client library in the solver code. The first call informs the visualization system about the following:

- Volume discretization -- the number of disjoint elements and their type and the number and size of structured blocks.
- Unsteady mode -- 'Steady-state', 'Grid', 'Data' or 'Structure Unsteady'.
- Visualization field variables -- the number, type, name and limits for each scalar, vector and threshold field.
- Programmed cuts -- Any non-planar geometric cuts defined for this case.

After a successful initialization, the process is entered into the **pV3** client group. For this application, the initialization call is inserted after the startup phase is complete.

The second call informs the visualization system of two things; (1) the simulation time for time-accurate solvers and (2) the solution has been updated and synchronized. In this application, this call must be placed at the bottom of the iterative loop. This is the mechanism that allows **pV3** to operate. Without the server running, every time the solution is updated and this call is executed, a check is made for any members in the

server group. If none is found, this call returns. When the visualization task starts, it enrolls in the server group, and waits for initialization messages from the clients (the solver). The next time the client goes through an iteration, and calls the **pV3** update routine, an initialization message is sent to the server. The server processes all initialization messages, figures out the number of clients and the visualization session begins. Each subsequent time the client goes through the loop and makes the **pV3** call, visualization state messages and tool requests are gathered. Then the appropriate data is calculated, collected and sent to the graphics workstation.

When the user is finished with the visualization, the server sends a termination message, leaves the server group, and exits. The clients receive the message and clean up any memory allocations used for the visualization. The **pV3** client update call reverts to looking for members in the server group.

The visualization system gets the solver's data via call-back routines supplied with the solver. These routines furnish details about the volume discretization, define the surfaces (partition as well as computational boundaries), the node locations and the requested scalar and vector fields. These are virtually the same call-backs used for a **Visual3** driver.

This software architecture has the following advantages:

- Control -- the solver is always in control, and the control mechanisms used during code development can be maintained.
- Minimum intrusion -- Only two lines need to be added (not including the call-back routines) to the solver code to allow co-processing visualization. Therefore, it can be removed easily for bug isolation.
- Steering -- The ability to steer the solution is provided via text strings passed from the server to all clients. The action is taken by a programmer supplied call-back that gets the string, interprets the command(s) and performs the requested action.

pV3's architecture does pose some difficulties to the code integrator. These involve informing the system on how to patch sub-domains together (for instantaneous streamlines and particle integrations) and in the definition of surfaces of interest (such as wing, body, tail, and etc.). Having the programmer define the surfaces allows **pV3** to automatically patch them together for the visualization. The programmer can lessen the sub-domain connectivity burden by trading-off computational efficiency for having **pV3** attempt to continue integrated paths anywhere throughout the domain.

References

1. Barth, T. and Linton, S., "An Unstructured Mesh Newton Solver for Compressible Fluid Flow and Its Parallel Implementation", AIAA Paper 95-0221.
2. Haimes, R., "**pV3**: A Distributed System for Large-scale Unsteady CFD Visualization", AIAA Paper 94-0321.
3. Haimes, R. and Giles, M., "**Visual3**: Interactive Unsteady Unstructured 3D Visualization", AIAA Paper 91-0794.