3.6 System Integration and Simulations

Team Coordinators: Mark Brandyberry and Robert Fiedler

Research Engineer: Courtlan McLay

Research Programmer: Johnny Norris

Undergraduate Students: Joseph Tortorelli and Rebecca Wingate

Overview

This year's efforts were devoted to extending the *Rocstar* version 3 rocket simulation package in several mission-critical areas: time zooming, surface propagation, and mesh modification. These capabilities were verified and applied to several large-scale simulations, including slumping in the Titan, complete propellant burn-out in a tactical motor, and substantial propellant burn-back in the RSRM. Significant effort was also devoted to devising how to reduce the number of lengthy simulations that must be run in order to quantify the effect of uncertainties in input physical parameter values on uncertainties in our simulation results.

Rocstar Description and Development

A diagram of the *Rocstar* 3 architecture is shown in Figure 3.6.1. A description of the specific modules that perform the functions written in each box is given below. Several new modules were fully integrated into *Rocstar* and several existing modules were significantly enhanced in order for us to apply our code to new problems of current scientific interest. The basic numerical methods and capabilities are summarized here. A number of these modules are described in more detail elsewhere in this document.

Input Dataset Preparation

On the left-hand side of Figure 3.6.1, the problem-definition tools and the physics solvers are represented by blue boxes (lighter blue for the solvers). CAD models and computational meshes are generated off-line using commercial packages plus preprocessors specific to each solver. The preprocessors are invoked by a comprehensive *Rocstar* dataset preparation tool (*Rocprep*) that largely automates the process of problem set-up.

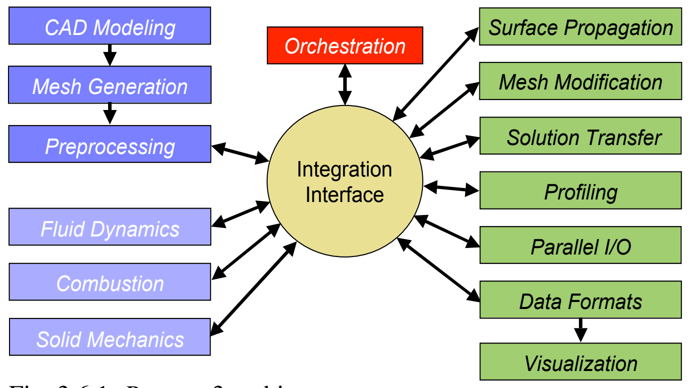

Fig. 3.6.1: *Rocstar* 3 architecture.

Physics Solvers

The three light blue boxes on the lower left in Fig. 3.6.1 represent the various general-purpose physics solvers that are available for use with *Rocstar*. The two fluid dynamics solvers are *Rocflu*, (unstructured meshes) and *Rocflo* (structured meshes). The basic algorithms in these codes were pioneered by Jameson. The fluid equations are formulated on moving meshes (referred to as the Arbitrary Lagrangian Eulerian, or ALE scheme) to handle geometrical changes during each time step, such as propellant burning and deformation, without the need to interpolate the solution onto the new mesh.

Rocflu uses unstructured tetrahedral or mixed tetrahedral/hexahedral/pyramid/prism mesh cells to handle complex geometries. An advantage of mixed meshes is the ability to employ hexahedral cells to provide high spatial resolution in boundary layers near physical surfaces. *Rocflu* relies on *Rocstar* to move and ensure the quality of its mesh. This finite-volume code employs a novel higher-order WENO-like approach, as well as the HLLC scheme to handle strong transients, including shocks. Time integration is accomplished via either the 3rd or 4th order explicit multistage Runge-Kutta time stepping algorithm, or by a Newton-Krylov-based implicit scheme that utilizes the PETSc library.

This year, our technique (called “time zooming” or “slow-time acceleration) for accelerating the longest time scales in physical problems that possess a wide range of well-separated time scales was integrated and verified in *Rocstar*. Time zooming differs from implicit time stepping by speeding up the longest time scales while preserving the behavior of the system on the shorter time scales, rather than ignoring the behavior on the short time scales. In principle, the two approaches may be combined if even greater speedups are required. Time zooming derives from an asymptotic analysis of the fluid equations to separate the long and short time scale behavior, while leaving the underlying form of the equations unchanged. In solid propellant rocket problems, the longest time scale is the time for the propellant to burn back significantly. Other important time scales, from shortest to longest, include the Courant time step limit for explicit schemes, turbulence evolution, droplet residence, and combustion-chamber filling. We assume that the dependence of the fluctuations of the fluid variables on the longest time scale may be neglected. Our analysis leads to the introduction of new source terms in the momentum and energy equations, plus modifications to the mass injection boundary condition at the burning propellant surface.

Rocflo uses either the central scheme or an upwind scheme involving Roe flux splitting on multi-block structured meshes. In addition to explicit Runge-Kutta time stepping, *Rocflo* can use the Dual Time Stepping algorithm to take steps longer than the Courant (CFL) limit. The mesh-smoothing algorithm used to maintain the quality of the multi-block structured mesh was substantially improved this year. Weighted Laplacian smoothing of the coordinates of block corners is first used to propagate the deformation among block corners. Transfinite interpolation is then used to distribute nodes along block edges while preserving the initial spacing. Finally, sets of elliptic PDEs derived from mesh quality measures, including the angles between adjacent edges, are solved, first for nodes on block faces, and then for interior 3-D volume nodes. This scheme works best when the block boundaries conform (not just the mesh cell faces at block boundaries, which must always conform).

Both fluid solvers can include a wide variety of physics options for simulating complex multiphase fluid flows. Non-ideal gases, chemical species (*Rocspecies*), droplets, smoke, and radiation (*Rocrad*), upgraded this year to a flux-limited diffusion approximation) may all be included with full, consistent, two-way coupling of mass, momentum, and energy between phases (*Rocinteract*). Burning aluminum droplets are treated by tracking Lagrangian superparticles, each representing many droplets (*Rocpart*), while smoke particles are evolved using the novel Equilibrium Eulerian method (*Rocsmoke*).

Primarily for use on structured meshes, three classes of turbulence models are available (*Rocturb*), including full Large Eddy Simulation (LES), Reynolds-Averaged Navier-Stokes (RANS), and Hybrid models (either LES with a near-wall model or Detached Eddy Simulation). Any combination of these turbulence models can be used in different regions of a single simulation, and various subgrid scale models and wall-layer models may be selected depending upon the flows being considered.

The rate of propellant deflagration is computed by one of three combustion modules. These physical models are one-dimensional (normal to the surface) in formulation, but are applied independently at each cell face on the burning propellant surface to obtain the local burn rate. The simplest model (*RocburnAPN*) assumes steady-state burning, for which the regression speed is proportional to the local gas pressure raised to the power “*n*”. Two dynamic burn rate models may also be selected. Both dynamic models solve a time-dependent heat conduction equation for the temperature profile near the propellant surface in order to capture ignition transients. One of these models (*RocburnZN*) is based on the Zeldovich-Novozhilov approach, while the other (*RocburnPY*) uses a simpler pyrolysis law.

Prior to ignition, propellant surface heating by igniter gases is calculated in *RocburnPY*. The propellant begins to burn when it reaches the specified critical temperature. Flame spreading is captured naturally as ignition on a given cell face contributes to propellant heating on nearby cell faces. A heat-flux look-up table computed by *Rocfire3D*, a detailed 3-D heterogeneous aluminized propellant combustion simulation code, can be used by *RocburnPY* to determine the local instantaneous burn rate. This enables very accurate reproduction of dynamic burning behavior at every cell face in a very large rocket at minimal additional computational cost.

RocburnAPN was recently extended to support multiple propellants at different axial locations in a rocket, which also allows us to include the effect of nozzle erosion. This capability is being used to improve Rocstar's prediction of the pressure history in the Attitude Control Motor (ACM), whose nozzle throat increases significantly during firing.

Rocstar includes two finite-element structural mechanics solvers, *Rocfrac* and *Rocsolid*. Both solvers feature an ALE formulation to account for the conversion of solid propellant into the gas phase. They handle large strains and rotations, can solve the 3-D heat conduction equation, and include a variety (wider this year) of element types and constitutive models.

Rocfrac has an explicit time integration scheme, and an implicit one that has been used in simulating helicopter blades. *Rocfrac* can include Cohesive Volumetric Finite Elements between ordinary elements to follow crack propagation.

Rocsolid has an implicit time integration scheme that uses the multigrid method and/or BiCGSTAB to solve the required linear systems efficiently in parallel. *Rocsolid* can utilize new nonlinear composite material models derived from state-of-the-art homogenization procedures to account for the evolution of the microstructure and include the effects of void formation and growth, damage (dewetting), and strain hardening

Integration Framework and CS Services

The Integration Interface (center of Fig. 3.6.1) is a library (API) called *Roccom*, which facilitates the exchange of data between different modules, including those written in different programming languages (C++, F90). By making a limited number of calls to *Roccom*, the physics applications gain access to a large number of useful Computer Science (CS) service components in our integration framework (column of green boxes on the right-hand side of Fig. 3.6.1).

In *Rocstar* the "modular" or "partitioned" approach to time stepping has been adopted, in which execution alternates between separate domain-specific physics solvers. This scheme has numerous advantages over the "monolithic" approach, in which all PDEs in the fully coupled problem are literally solved simultaneously as a single large linear system. The modular approach facilitates software development and maintenance, incorporation of legacy solvers, interdisciplinary collaboration, etc. Time-accurate solutions to fully coupled problems may be obtained using the modular approach, provided the exchange of interface quantities between modules is performed frequently enough.

The orchestration module called *Rocman* (red box in Fig. 3.6.1) controls the execution of the physics applications, including initialization, coupled time stepping, writing output files, and stopping criteria. *Rocman* also enforces interface jump conditions (derived from conservation of mass, momentum, and energy) specific to the particular coupled problem under consideration. We do not require the individual physics solvers to be customized to address any specific coupled problem.

One may choose from several system time stepping schemes, including the Simple Staggered Scheme with optional Predictor-Corrector (P-C) iterations and the Improved Staggered Scheme of Farhat. In the Simple Staggered scheme, the solid, fluid, and combustion solvers each execute one system time step (which may comprise several internal time steps for explicit solvers) in a round-robin fashion. A typical system time step for a large booster is 20 microseconds. After each solver completes its system time step, it exchanges interface data with the other two solvers, which can use that information to improve solution accuracy as they take their system time step. To further improve accuracy and stability, corrector iterations are executed, in which the system time step is repeated until the L_2 norms of the interface quantities change by amounts that are less than a user-specified tolerance. If this tolerance is sufficiently stringent, the overall accuracy of the time stepping scheme is equal to that of the lowest order physics solver. A more stable, accurate, and efficient fully-implicit coupling scheme is under development, but is not required to obtain accurate solutions to whole-system rocket problems, where the solid density is much greater than that of the gas.

The solution transfer module (*Rocface*) enables the physics applications to exchange interface quantities across non-matching meshes, which is essential to solving coupled fluid/structure interaction problems. By construction, the interpolation scheme exactly conserves mass, momentum, and energy at the interface. The scheme makes use of an overlay mesh, which is a common refinement of the two meshes on either side of the interface. Each subdivision of the overlay mesh lies entirely within a cell face in both surface meshes. Thus, the overlay mesh enables accurate integration of quantities that depend on the shape functions of the two meshes. Interpolation errors are minimized in the least squares sense, leading to a scheme that has been demonstrated to be many times more accurate than other recently published methods. Although minimizing interpolation errors involves solving linear systems, this procedure consumes only a small fraction of the run time for a typical *Rocstar* simulation.

The surface propagation module (*Rocprop*) computes the motion of the propellant surface as it regresses due to burning. *Rocprop* can be used in coupled simulations as well as fluids-only or solids-only calculations. *Rocprop* includes a new, robust, accurate, and efficient surface propagation scheme called the Face-Offsetting Method (FOM), which is based on a generalized Huygens principle. This method operates directly on a Lagrangian surface mesh, without requiring an Eulerian volume mesh, as do typical “level-set” methods. Unlike traditional Lagrangian methods, which move each vertex directly along an approximate normal or user-specified direction, FOM propagates faces and then reconstructs vertices through an eigenvalue analysis performed locally at each vertex to resolve the normal and tangential motion of the interface simultaneously. The method also includes improved techniques for detecting and maintaining surface features, which is a crucial capability in determining the correct orientation of the surface normal vector at corner and edge nodes. This enables FOM to maintain the smoothness and integrity of the surface as it evolves, even in the presence of singularities and large curvatures, and to smooth the surface mesh by sliding nodes only in directions tangent to the surface.

Constraints on surface motion are now supported in *Rocprop* (and temporarily in *Rocflu*) to allow fluid-only simulations that include regressing propellant. The constraints prevent the propellant from burning beyond the rocket case. Cylindrical cases with ellipsoidal or hemispherical ends are currently supported. Imposing more general constraints based on an input mesh or CAD model is under development.

The mesh modification schemes in *Rocstar* are employed at two levels. Mesh smoothing (moving mesh vertices without changing their number) for unstructured meshes is accomplished in the *Rocmop* module through subroutine calls to the serial Mesquite package developed at Sandia National Laboratory. Each partition calls Mesquite concurrently, sending both real and ghost nodes. Mesquite smoothes only the interior nodes of these mesh partitions, so including the ghost nodes is essential to maintaining mesh quality. After Mesquite smoothes all partitions, the coordinates of vertices shared by multiple partitions are averaged to ensure that the meshes still match at partition boundaries. It is possible (but not usually necessary) to call Mesquite multiple times to alleviate any impact on mesh quality due to averaging shared node coordinates. Because the evolution equations in our solvers are formulated on moving grids, no solution transfer is required after mesh smoothing, although the amount that the mesh can change locally per call is limited by a Courant-like stability criterion. Support for non-tetrahedral element types is now included in *Rocmop*. If the geometry changes occur at a sufficiently slow rate, we may perform mesh smoothing only once every 3rd fluid time step, saving substantial CPU time compared to smoothing every step.

If smoothing cannot maintain acceptable mesh quality, either local mesh repair or global remeshing can be performed using tools from Simmetrix, a company spun off from Professor Mark Shephard's group at Rensselaer Polytechnic Institute. We currently assemble the partitioned mesh on a single processor and use it to create a new mesh by means of the serial version of Simmetrix. The surface mesh spacing can be preserved during remeshing, whether or not the user chooses to preserve the surface mesh. The new volume mesh is constructed using the (remeshed) surface mesh, with surface-element sizes and gradients in volume element size controlled by Simmetrix input parameters. Parallel implementations of the serial mesh-repair and remeshing steps are under development. The *Rocrem* module currently enables serial remeshing. It also performs parallel solution transfer to the new mesh, and parallel generation of all input files required to restart a simulation through calls to the newly developed *Roctail* module (described below). These parallel steps can be performed even on platforms that are not supported by Simmetrix, such as the IBM SP (e.g., ASC Purple). We can also use Simmetrix tools to perform local mesh repair, through the new *Rocrep* module (under development), which operates only on selected mesh partitions. We are investigating ways to save wall clock time by taking advantage of the fact that much of a repaired mesh remains unchanged.

The new *Roctail* module greatly simplifies the use of third-party meshing and partitioning tools by providing translation and interface services to other *Rocstar* modules. *Roctail* provides a neutral mesh and data representation, which can be translated to *Rocstar* or physics solver native data formats. *Roctail* also provides data services such as attribute extraction, common refinement-based intermesh data transfers, surface merging, and solver native initialization. These services are available both through an offline library and at runtime during a *Rocstar* simulation.

Thanks to *Roctail* and a number of extensions to *Rocman* and the physics solvers, in many situations remeshing can be applied automatically many times during a simulation without user intervention. A new "warm" restart capability enables *Rocstar* to remesh without terminating the simulation. Automatic remeshing currently can be triggered by the fluid time step size. Improving the implementation of warm restarts and adding more triggering criteria, including one based on estimation of truncation errors, are ongoing projects.

Rocstar automatically collects performance data for functions registered with *Roccom*, including physics application solution update times, data transfer times, output dump write times, etc. Profiling at the subroutine, loop, or statement levels can be performed by inserting into the source code a small number of low-overhead calls to the *Rocprof* module. *Rocprof* was redesigned this year for the current version of *Rocstar*. It is designed as a light-weight, stand-alone CS service module rather than a tightly integrated sub-module of *Roccom*. As a new CS service module, *Rocprof* compliments *Roccom*'s automatic high-level profiling by providing enhanced profiling services to any code in the *Rocstar* suite. *Rocprof*'s services are provided through native and standard MPI_PCONTROL interfaces, allowing use by any serial or parallel application. The *Rocprof* suite now also includes post-processing tools with which to archive, report, and analyze multiple sets of parallel and serial performance data. *Rocprof*'s new design consolidates the stand-alone and integrated code bases, streamlining further development. Support for MacOS X 10.3 and 10.4 has been added. Charm++ interoperability has been enhanced and tested. PAPI support has been reinstated.

We have put *Rocprof* to good use in profiling and tuning *Rocstar* components. Sub-module profiling has enabled us to identify and eliminate many performance bottlenecks in *Rocstar* physics and service modules. Ongoing performance analysis and tuning efforts of *Rocflu* have yielded substantial performance boosts of up to factors of 4 for critical code sections. We have been collaborating with Livermore Computing in the *Rocflu* tuning effort. *Rocprof* has also identified several bottlenecks in *Rocstar*'s mesh smoothing service module, *Rocmop*. Subsequent elimination of these bottlenecks has reduced overall smoothing overhead by a factor of 2. *Rocprof* is currently being used as a development tool for new features of the fluid solver framework.

Asynchronous Parallel I/O can be performed using *Rocpanda*. *Rocpanda* designates a user-specified number of processes as I/O servers, which collect data in the form of MPI messages from the compute processes, combine the data, and write it to a manageable number of disk files in the desired format in the background as the simulation continues.

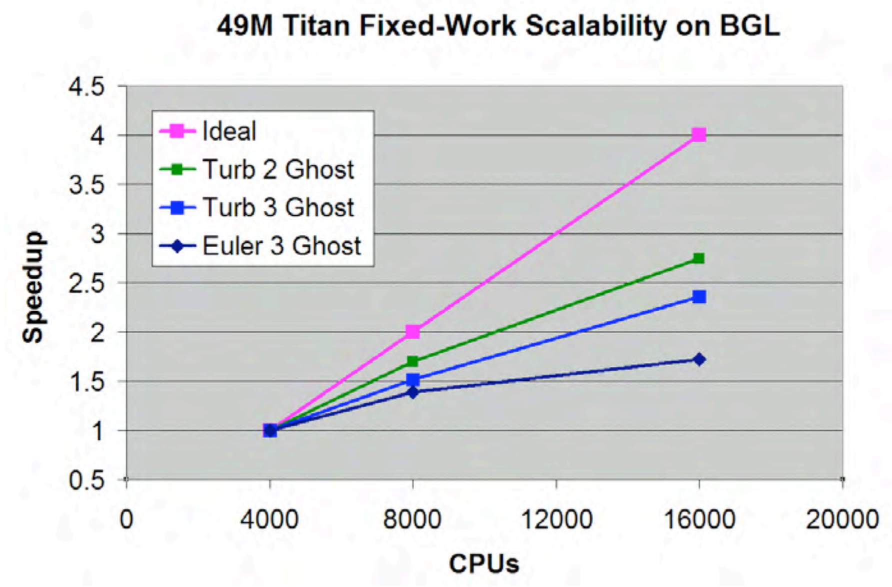

Fig. 3.6.4: Scalability of *Rocflo* on BGL for a Titan mesh with 49 Million cells.

All modules in *Rocstar* use MPI (Message Passing Interface) to pass messages between processes running in parallel. The modules are compatible with AMPI (part of the Charm++ framework), an implementation of MPI that treats processes as user-level threads. There are two key benefits of AMPI for *Rocstar*: 1) the AMPI processes are "virtual" so that they can run on any number of actual CPUs, and 2) the virtual processes can be migrated from one CPU to another for dynamic load balancing. In performing large rocket simulations, we have used the first of these two features extensively to utilize available computational resources (fewer processors available than the number of partitions). Dynamic load balancing becomes important after the meshes have been refined or coarsened in some partition, due to either geometrical changes (e.g., propellant burning and deformation) or solution-based mesh adaptation.

Scalability on BGL

The BlueGene/Light machine at LLNL (BGL) was made available briefly last winter for use by the ASAP Centers before it was moved permanently to a classified network. We explored the limits of *Rocflo*'s scalability on this machine using a structured mesh for the Titan IV booster with 49 Million elements. We reduced the memory usage of *Rocflo* during initialization by several orders of magnitude for meshes with large numbers of partitions. Data structures in *Rocflo* that had been included for convenience to allow every partition to communicate with potentially every other partition were replaced by much shorter lists of only those partitions with which a given partition needs to exchange information. *Rocstar*'s IO strategies and modules were modified to handle the extremely large numbers of files in datasets of this size and quirks of the LUSTRE filesystem as implemented on BGL. Parallel UNIX system tools were created to rework *Rocstar* datasets to make running on BGL possible.

Figure 3.6.4 shows speedups for the 49 million element mesh on 4000, 8000, and 16,000 processors (i.e., 12,000, 6,000, and 3,000 elements per processor, respectively). Since the number of elements per CPU is quite small on 16,000 CPUs, speedups are considerably less than ideal, particularly for an inviscid (Euler) fluid, for which the amount of computational work per time step is relatively small. Including viscosity and turbulence (Turb 2 Ghost) increases the ratio of computational work and communication, and therefore improves scalability significantly. Use of a subscale energy model that requires a larger stencil (Turb 3 Ghost) increases communication without much additional computational work, and therefore scalability is somewhat reduced. In production runs, we normally use 8000 or more elements per CPU on every platform to ensure good scalability.

We will continue to optimize *Rocstar* on the upgraded 3000-node BlueGene system at San Diego Supercomputing Center and other platforms with small amounts of memory per CPU.

Simulations

Star-Aft Motor with Time Zooming

We tested the Face-Offsetting surface propagation algorithm, mesh motion constraints at the rocket case, and the time zooming technique, using a tactical rocket motor with a star shaped propellant profile in the aft end (Star-Aft" motor). Figure 3.6.5 shows the propellant configuration for a time-zoomed run at 4 successive times during firing, progressing from upper left to upper right, and then from lower left to lower right. The red color indicates burning propellant, while purple indicates non-burning regions. The mesh was smoothed every time step, and remeshing was required 23 times to maintain good mesh quality despite the significant geometrical changes that occur, especially as the star fin tips change from rounded surfaces to sharp ridges, and as the star slots encounter the case.

To quantify solution accuracy while time zooming, we performed a number of simulations of the Star-Aft motor with different slow-time acceleration (zoom) factors *Z*. Figure 3.6.6 shows the head-end pressure histories. Time zooming was switched on in these simulations at 50ms, when the quasi-steady state was reached. Note that even for a zoom factor of 64, the error in the head-end pressure (compared to the case without time zooming, Z = 1) is never more than a few percent. Thus, the magnitude of errors due to time zooming with Z = 64 are smaller than the truncation errors on this relatively coarse mesh. The pressure histories in Fig. 3.6.6 for smaller zoom factors do not extend as far as those for large zoom factors simply because of the computational cost of these simulations. Nevertheless, we plan to extend all of these runs to complete burn out.

Reusable Solid Rocket Motor with Time Zooming

We performed *Rocflu* fluid-only simulations with significant propellant burn-back in the Space Shuttle Reusable Solid Rocket Motor (RSRM) this year for the first time using our time-zooming technique. The zoom factor was set to a large (1024) value once the ignition transient had passed and the quasi-steady burn phase was reached. In order to test our surface propagation and remeshing capabilities, the burn rate was set to a constant value (1 cm/s). Subsequent simulations (in progress) adopt a burn rate that depends on the local pressure, and will be used to

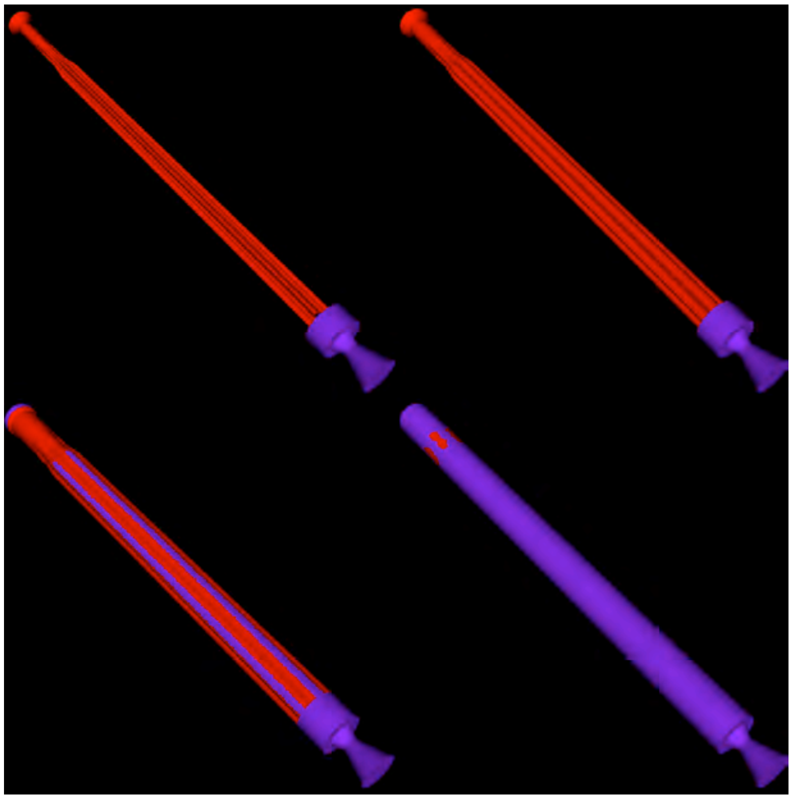
Fig. 3.6.5: Propellant burn back in Star-Aft motor. Red indicates burning propellant surface.

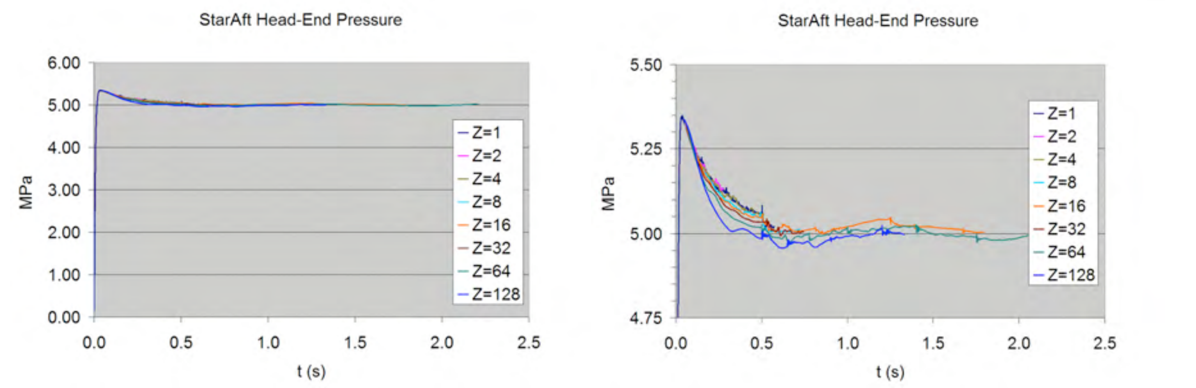

Fig. 3.6.6: Head-end pressure histories for Star-Aft motor for different zoom factors. Right panel shows same data on scale with narrower pressure range. Small pressure spikes are due to solution transfer truncation errors when remeshing.

determine the dependence of the error on the zoom factor.

The model geometry includes many details, such as stress-relief features at the aft ends of the 11 star-fin slots, correct inhibitor radii in each joint slot, and a "submerged" nozzle. The unstructured fluid mesh contains 4.5 million tetrahedral elements and is smoothed by calling Mesquite every few fluid time steps. Remeshing is triggered automatically whenever the fluids time step falls below a threshold value (10^{-7} s). Since the zoom factor is so high, the propellant regresses quite rapidly, and remeshing is triggered so frequently that only a fraction of the wall clock time is spent solving the fluid equations. However, such large zoom factors lead to pressure histories that differ markedly from the experimental data. Simulations with zoom factors small enough to produce reasonably accurate pressure results will perform far more fluid time steps between instances of remeshing, and therefore the ratio of the fluid solver time and the remeshing time will be much more reasonable.

Figure 3.6.7 shows the propellant configuration at 10 ms (upper left panel), 16 ms (upper right), 22 ms (lower left) and 29 ms (lower right). Note that the motion of the surface is constrained by the cylindrical case for most of the rocket's length, plus an ellipsoidal dome at the head end. Surface nodes slide along the dome beginning at early times, and the image at 29 ms shows that the star slot tips have reached the cylindrical portion of the case. By this time, the burning propellant (colored orange) in the center and aft joint slots has nearly reached the case just forward of the forward-facing inhibitors. Titan IV Propel-lant Slumping

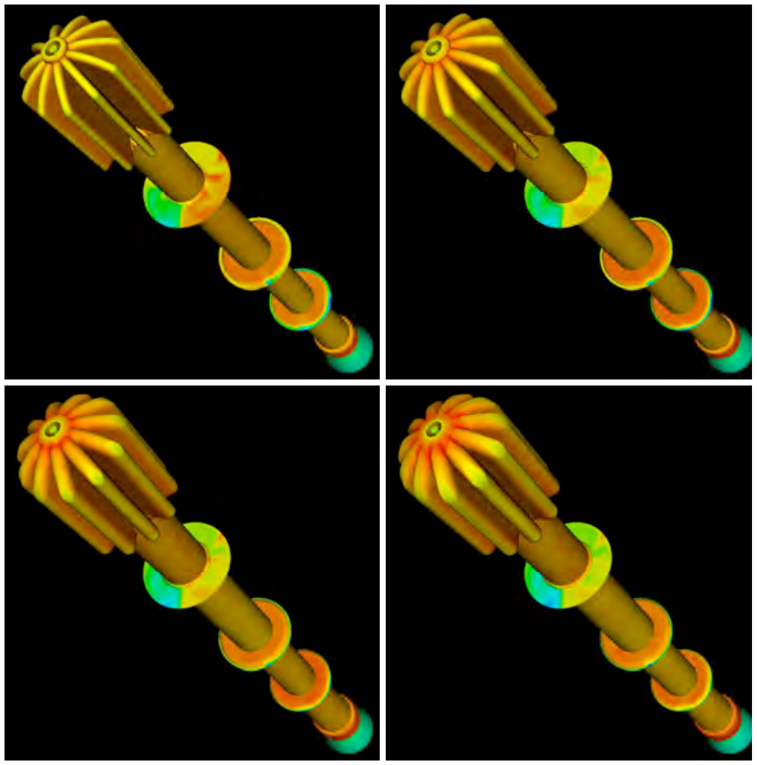

Fig. 3.6.7: Propellant configuration at 4 different times in RSRM. Color scale indicates gas temperature over narrow range.

We used *Rocstar* to improve upon our previous 3-D simulations of propellant slumping in the Titan IV SRMU Prequalification Motor #1 (PQM-1). We are investigating an aeroelastic effect that occurs at joint slots where, for manufacturing reasons, there is a smaller propellant bore on the aft side. The thermal pressure is considerably higher inside the slot than it is just downstream of the slot, because the gas speed is much higher there. This uneven load distribution tends to pull the propellant toward the axis of the rocket on the aft side of joint slot, possibly to the point of choking the flow of exhaust gasses. This effect caused the destruction of this motor on the test stand in 1991.

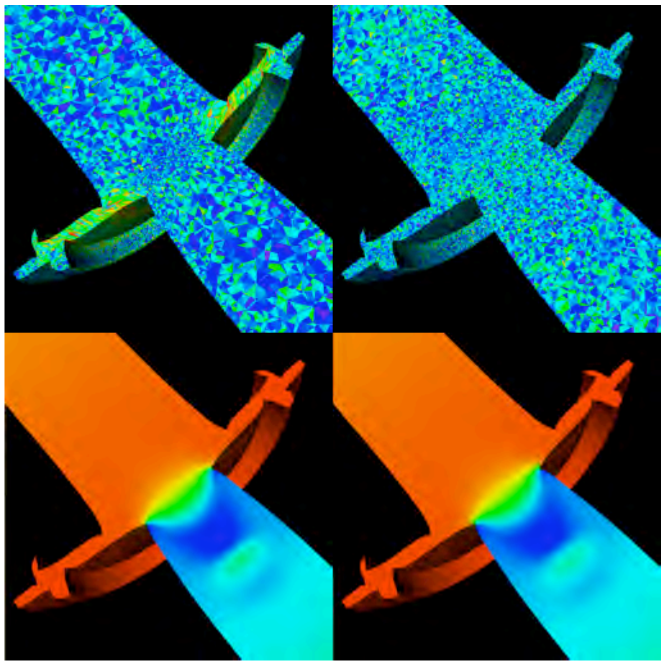

Fig. 3.6.8: Top: maximum dihedral angle in each mesh element for simulation with smoothing only (left) and one with smoothing plus remeshing (right). Bottom: Pressure field transferred from old mesh (left) to new mesh (right).

A new non-linear viscoelastic propellant material model, developed by Sofronis, includes the effect of void formation and growth, damage (dewetting), and strain hardening. This model was recently implemented in our implicit structural dynamics solver *Rocsolid*. We performed new fully-coupled simulations of the Titan using *Rocflu* and *Rocsolid*. We found that for any material model, the propellant deforms much earlier than it did in the actual rocket. We are now aware that the bore diameters are rather different in our model than they are in the actual motor after viewing a presentation by ATK. The "overhang" at the center joint slot is exaggerated by about a factor of 2 in our model, which is due to the fact that we estimated our geometry from a paper in the open literature. However, the simulations we performed do serve as useful tests of remeshing.

Figure 3.6.8 (top two panels) shows maximum dihedral angles in (cut) mesh cells near the center joint slot in two different simulations. In the simulation corresponding to the left-hand panel, the mesh was smoothed by Mesquite, but not remeshed. In the simulation corresponding to the right-hand panel, the mesh was both smoothed and, when necessary as determined by the fluid time step size, remeshed using Simmetrix. Note that the entire rocket was modeled in 3-D. Some mesh blocks are not shown, allowing a view of the mesh in the slot. This portion of the rocket is nearly axisymmetric. There are initially four tetrahedral elements across the joint slot, and as the slot opens due to the load on the propellant, in the absence of remeshing these slot elements are stretched several times their initial length, primarily in the axial direction.

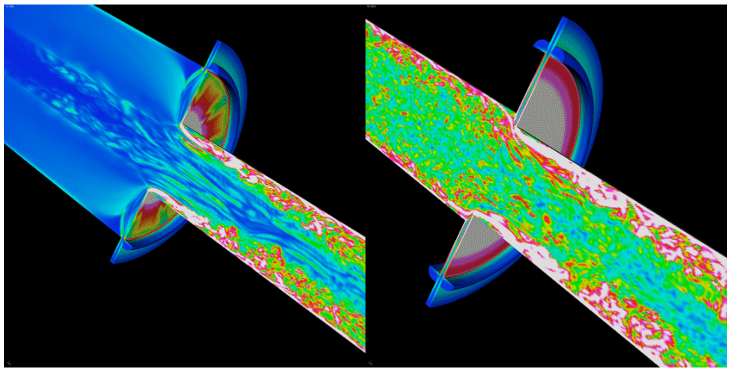

Fig. 3.6.9: Top: maximum dihedral angle for simulation with smoothing only (left) and smoothing plus remeshing (right). Bottom: Pressure field transferred from old mesh (left) to new mesh (right).

In this situation, Mesquite by itself cannot maintain acceptably high mesh quality. However, in the run with remeshing enabled, the mesh quality remains high at all times, and truncation errors are much smaller. Errors introduced by solution transfer are on the order of about 1 percent or less, as seen in Figure 3.6.8 (bottom two panels).

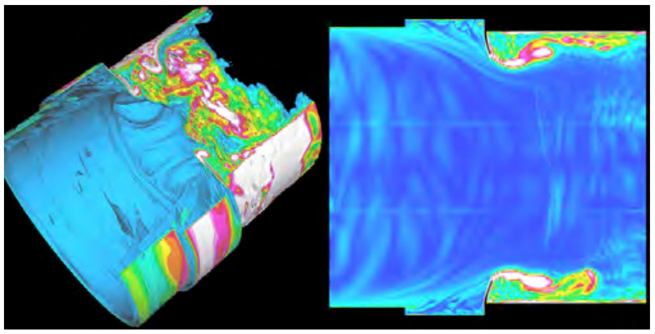
Fig. 3.6.11: Vorticity magnitude isosurfaces (left) and color scale plot (right) in a section of the RSRM near center joint slot 100 seconds after ignition.

RSRM and Titan IV with Turbulence

We simulated turbulent flows in both the RSRM and the Titan IV SRMU PQM-1 using structured meshes of varying resolution ranging from a total of 1.5 million to 21 million cells. In each of these *Rocflo*-only simulations, we evolved the system for a few seconds of physical problem time while neglecting propellant regression, in order to study pressure fluctuations in a statistically steady turbulent flow. Mass was nevertheless injected at the burning propellant surface according to the pressure-dependent burn rate law. We used a Large Eddy Simulation (LES) turbulence model with the fixed or the dynamic Smagorinsky subgrid scale model for the turbulent stress tensor in these calculations. We have recently demonstrated that the dynamic Smagorinsky model most accurately predicts turbulence quantifies for injection driven flows in a duct of square cross section and a flared nozzle.

Figure 3.6.9 shows the vorticity magnitude in the Titan near the head end (left panel) and near the center joint slot (right panel). Note that vortex sheets are shed wherever the flow goes around geometrical obstructions or strong cross flows at uninhibited joint slots. These vortex sheets soon develop into fully turbulent flows. It was not necessary to inject turbulent fluctuations at the burning propellant surface in order to produce a turbulent flow, even in the RSRM, in which the overhangs at joint slots are much smaller than they are in the Titan.

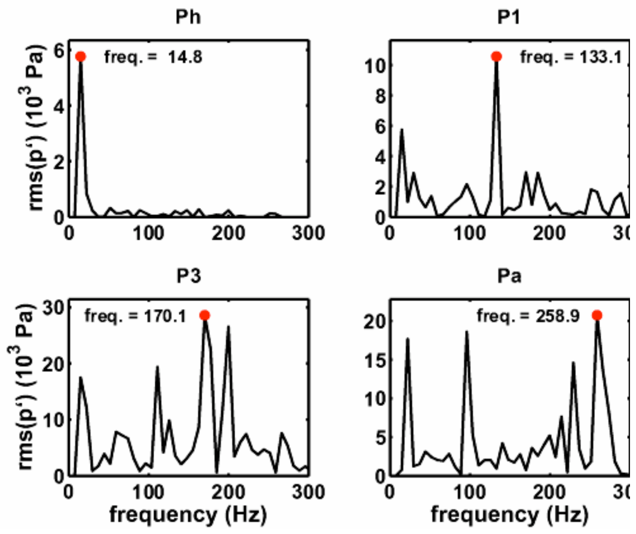

Fig. 3.6.10: Pressure fluctuation amplitude as function of frequency at four different axial locations of near-wall probes.

Figure 3.6.10 shows results from a Fourier analysis of pressure fluctuations recorded by probes in the RSRM near the burning surface at 4 different axial locations: (h) head-end, (1) just aft of the forward joint slot, (3) just aft of the aft joint slot, and (a) aft end. The probe at the head end picks up mainly the fundamental longitudinal mode for sound waves, as do all probes along the axis of the rocket. However, near the propellant surface, downstream from joint slots and at the aft end, the dominant pressure fluctuation magnitudes occur at frequencies corresponding to vortex shedding rather than to acoustic waves. This is important information for rocket designers, since pressure fluctuations can couple with combustion instabilities.

RSRM Flexible Inhibitor with Turb ulence

We performed fully-coupled *Rocstar* simulations of the gas flow in the vicinity of a flexible inhibitor protruding into the combustion chamber of the RSRM, including LES turbulence (dynamic Smagorinsky model). The geometry matches the RSRM at 100 seconds after ignition, when the inhibitor protrudes 25 to 30 cm into the combustion chamber. We modeled a 4-meter long section of the rocket near the center joint slot in these *Rocflo/Rocsolid* simulations, using 3 million structured fluid elements. The inflow conditions make use of velocity profiles computed in an older full RSRM simulation. The turbulence fluctuations introduced at the inflow boundary upstream of the inhibitor are determined by recycling the fluctuation components at some distance downstream. Finally, a partially non-reflecting mixed subsonic/supersonic boundary condition is imposed at the outflow boundary, where the static pressure is held at 8.3 MPa.

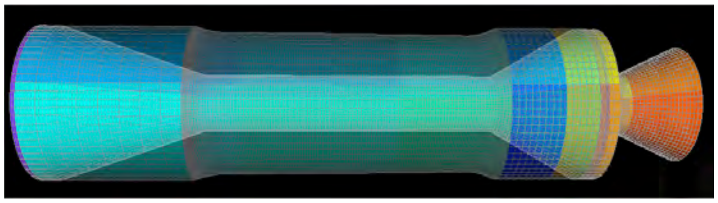
Fig. 3.6.13: Attitude Control Motor simulation fluid domain.

To avoid contact of the inhibitor with the burning propellant surface downstream of the joint slot during the strong shock wave due to the initial conditions adopted in this simulation, the Young's modulus of the inhibitor was set artificially high (by a factor of 100) until after the flow begins to settle down. We used a non-linear neoHookean constitutive model for both the inhibitor and propellant.

Figure 3.6.11 shows vorticity magnitude isosurfaces (left) and a color scale plot on a slice down the axis (right) after a turbulent flow has been established in a simulation with a flexible inhibitor. The inhibitor swings back and forth chaotically due to gas pressure forces, causing the vortex shedding frequency to fall and rise, introducing pressure fluctuations that could drive combustion instabilities in the motor.

We performed a Fourier analysis of pressure fluctuations recorded by probes (as described above for the full RSRM), but in this case we found that the fundamental mode of the 4-meter section dominates all other frequencies. While it might be possible by various means to reduce the effect on the fluctuation spectrum of the artificial domain boundaries we imposed in this simulation, the most reliable way to improve our results would be to simulate the entire rocket, perhaps with a somewhat lower spatial resolution away from the inhibitors.

Figure 3.6.12 shows a slice of the 3-D the structured grid in the vicinity of the inhibitor at a time when it is bent back by a relatively large angle. The different colors indicate different mesh blocks. Note that the grid lines remain smooth and nearly orthogonal, even near the rear edge of the inhibitor.

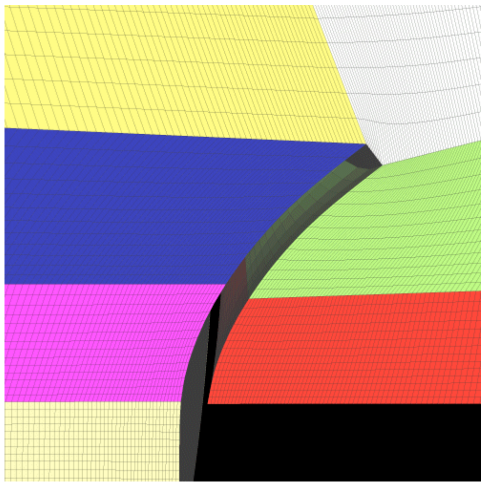
Fig. 3.6.12: Slice of 3-D structured mesh in region very close to the inhibitor at time when angle of deflection is relatively large. PDE solver maintains optimal mesh quality at all times.

RSRM

Under a contract with ATK, CSAR is simulating the next generation reusable solid rocket motor for NASA's Aries 1 launch vehicle prior to fabrication. We have preformed a mesh resolution study and computed the quasi-steady inviscid flow at 0.6 seconds after ignition. The head-end pressure in a run with 3.9 million tetrahedral elements differs from the pressure in a run with 2.5 millions tetrahedral elements by less than 1 percent. Our results are in good agreement with the values obtained by an analysis performed at ATK.

Uncertainty Quantification

A method for estimating the parameter uncertainty in CSAR *Rocstar* results has been developed and successfully applied to a typical solid rocket problem using a small number of *Rocstar* simulations. The small Attitude Control Motor (ACM) shown in Figure 3.6.13 (fluid-domain shown; solid colors depict the initial configuration, while translucent colors indicate the 95% burned-back configuration) has been simulated with *Rocstar* from ignition through almost complete burnout. Uncertainties in 5 input parameters described with probability distributions have been modeled using a set of 10 *Rocstar* runs to produce results such as the head-end time-pressure uncertainty envelope shown in Figure 3.6.14. The solid curves show the statistical envelope of pressures that can be expected, given the parameter distributions used in the simulation. The symbols in the plot show the three available experimental time-pressure traces plotted against the *Rocstar* uncertainty results.

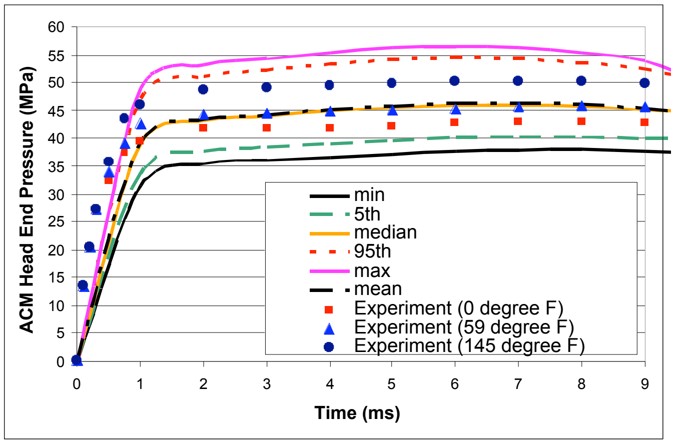

Fig. 3.6.14: Attitude control motor head end pressure time history; *Rocstar* simulation uncertainty versus experimental results.

The number of *Rocstar* simulations required to obtain the pressure-history envelope is reduced by using a clustering technique in combination with stratified Monte Carlo Sampling. The ten simulations performed to generate the results in Figure 3.6.14 are a close approximation to using 500 *Rocstar* runs, chosen by grouping together (clustering) samples that would provide similar *Rocstar* results.

The experimental results in Figure 3.6.14 show excellent agreement with the general uncertainty range calculated by the *Rocstar* runs. However, visual comparison is not sufficient for performing V&V with uncertainty, and thus a statistical comparison of the *Rocstar* distribution against the experimental results was performed. Using a statistical means test at each millisecond of the simulation, figure 3.6.15 shows the significance values calculated at each millisecond for comparing the distribution from *Rocstar* against the experimental distribution assumed from the 3-point sample at each millisecond of time. A significance above 0.05 indicates that the *Rocstar* mean and the experimental mean are statistically indistinguishable. Thus, the visual comparison agreement between the *Rocstar* and experimental results is verified statistically for times 1 ms and greater.

Research into these and other methods for uncertainty characterization and validation with uncertainty are continuing at CSAR. Investigation of methods for including model-form uncertainties and error estimates in the results along with propagated parameter uncertainties are also ongoing.

Software Engineering Advances

For the majority of its existence, CSAR has configuration-managed its software code base using CVS version control software. Approximately 4 years ago, an automated build system was implemented to build *Rocstar* on multiple target platforms, with automated web and e-mail based reporting of nightly builds to *Rocstar* developers. This automated build system has become instrumental in keeping *Rocstar* in a state that will always build on most platforms, verified nightly. A recent analysis has estimated that approximately 55% of the nightly builds complete successfully with no build errors over a 2-year period. The other 45% generated one or more build errors of which the relevant developers were immediately notified, and the problem was promptly corrected.

Until this year, attempts at automated regression testing for *Rocstar* had not been as successful. Running many regression tests nightly on the production systems that were available to CSAR became untenable beyond a few jobs per night, since the target machines were generally very busy. This fact has led to the construction of an entirely new regression-testing platform dedicated to *Rocstar*.

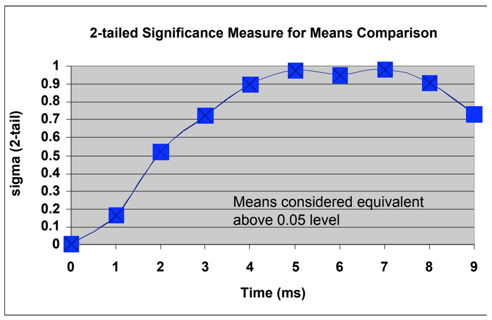

Figure 3.6.15. *Rocstar* uncertainty distribution with experimental distributions over time.

The Hilbert build and test cluster has been constructed from used components from an older departmental system. The cluster has 32 computational processors, a high-speed Myrinet interconnect, and a 4-processor head node. It is a substantial resource that has many of the features of other Linux-based clusters on which *Rocstar* must run.

A new Java-based build and regression testing system has been constructed to run on the Hilbert cluster. It is designed to build *Rocstar* in numerous configurations nightly, and then to run a series of regression tests on versions of *Rocstar* built that night. Currently, 32 tests across six different versions of *Rocstar* are being run nightly.

For each regression test that is run, completion of the test is confirmed first, and then the full output file set for that run is compared against a “gold standard” set of results for the test using a comparison tool called *Rocdiff*. The files are compared on a grid-point and individual output variable basis, and measures such as the maximum difference and mean squared error are calculated. The testing tool then uses a set of user-defined comparison criteria to validate that the regression test does, indeed, produce effectively the same results as the gold standard.

Further regression test problems are being added to the regression suite. A matrix of test coverage based on code functionality exercised for each test problem has been assembled. This matrix is used as the basis for selection of new tests to ensure that they cover new portions of *Rocstar*.