3.6 System Integration

Team Coordinators: Mark Brandyberry and Robert Fiedler

Simulating solid propellant rocket motors requires solving an extremely complex, fully coupled multidisciplinary physics problem that spans a wide range of length and time scales. We have taken a staged approach in developing an integrated whole-system rocket simulation package, beginning with relatively simple physical models, geometries, and coupling in our first-generation code (GEN1), and progressing toward more detailed physics, geometries, and interactions in our second-generation code (GEN2).

GEN1

Implementation of GEN1 version 2.0—the final version of our simplified, fully-coupled rocket simulation code—was completed in May 2001. The physics modules in GEN1 include *Rocflo*, an explicit, finite-volume, Navier-Stokes fluid dynamics solver; *Rocsolid*, an implicit, multigrid, finite element, structural dynamics solver, and *Rocburn*, an unsteady burn rate code. The GEN1 physics modules are written in Fortran 90, are fully parallel, and use the MPI library to pass messages between processes.

The fluids and structures modules utilize an Arbitrary Lagrangian Eulerian (ALE) formulation to follow moving boundaries and combustion interfaces. The equations of motion for the structural response of the case of a regressing interface result in a non-symmetrical linear system, which is solved using the BiCGSTAB method. The multigrid solver in *Rocsolid* is retained as a more efficient option for problems without significant propellant burnback. Nonlinear kinematics (large displacements) have recently been incorporated in *Rocsolid* to enable simulation of propellant slumping in the Titan IV SRMU and the deformation of flexible inhibitors.

Rocburn computes the dynamic burn rate given the local pressure at each fluid cell face on the propellant surface. It is based on the Zeldovich-Novozhilov phenomenological model. The gas phase is treated as quasi-steady, and the 1-D unsteady heat conduction equation in the condensed phase is solved to obtain the temperature profile in a thin layer near the surface. Compared to the steady burn rate, the dynamic burn rate is enhanced during the phase of rapid pressurization shortly after ignition due to the transient storage of energy at the propellant surface. This mechanism, rather than "erosive burning," is responsible for the pressure overshoot seen in some motor configurations. For example, the ignition spike computed in a simulation of Tactical Motor #13 of Blomshield, et al., is in qualitative agreement with the experimental data (Figure 3.6.2), although an accurate ignition and flame spread model must be included to closely match the experiment.

Transfer of data across fluid/structure boundaries is accomplished by *Rocface* v1.0, which performs mesh association and interpolates quantities in a globally conservative manner, using the algorithm described by

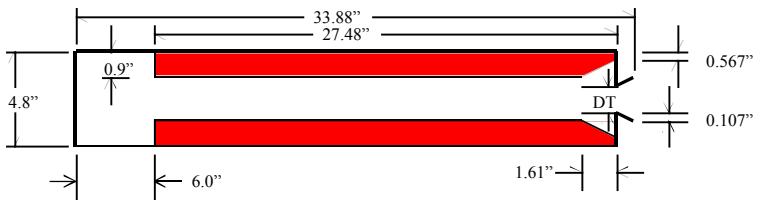

Fig. 3.6.1: Schematic of "Tactical Motor #13" used for GEN1 validation.

Farhat, et al. This implementation is restricted to matching meshes for the case of regressing combustion interfaces.

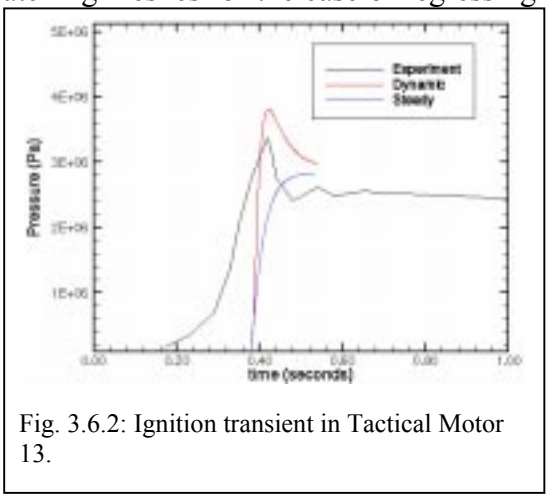

Fig. 3.6.2: Ignition transient in Tactical Motor 13.

The combustion interface is treated as a thin layer where appropriate jump conditions derived from conservation of mass and momentum are applied to couple the fluids and solids solutions in space.

Temporal coupling is accomplished by a predictor-corrector time stepping scheme. In the predictor cycle, the fluids code takes a number of short explicit time steps using an estimate of the motion of the solid from the previous step, and passes its updated interface pressure values to the solids code. Then the solids code takes one large implicit step to catch up with the fluids code, and passes the updated surface motion to the fluids code. In the corrector cycle, the fluids code repeats its steps with an improved estimate of the motion of the solid, and the solids code repeats its step with an improved estimate of the surface pressure. The corrector cycle is repeated until the changes in each interface quantity from one iteration to the next are below prescribed tolerances.

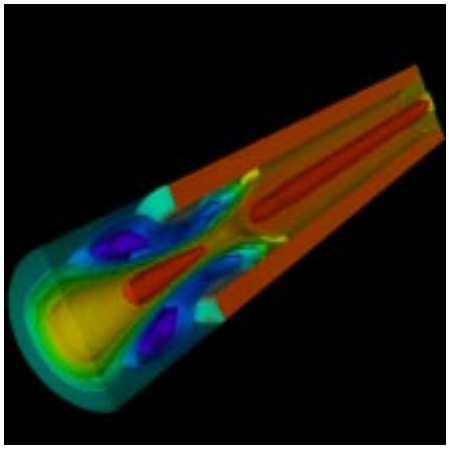
Fig. 3.6.3: GEN1 simulation of Tactical Motor #13 shows temperature in fluid and stress in solid propellant.

The scalability of the GEN1 code on blue horizon, an IBM SP at SDSC, is shown in Figure 3.6.4 for both the multigrid and BiCGSTAB algorithms. This test problem has simple cylindrical geometry and the computational workload and communication per processor is held constant so that the wall clock execution time on any number of processors should be constant. Performance data are shown for two grid sizes. The BiCGSTAB algorithm is slower but just as scalable as the multigrid solver on up to 256 processors. The tail-off in scalability beyond 512 processors is entirely due to the fluids solver (which has since been optimized to eliminate the unnecessary searching that was impacting scalability on larger numbers of processors).

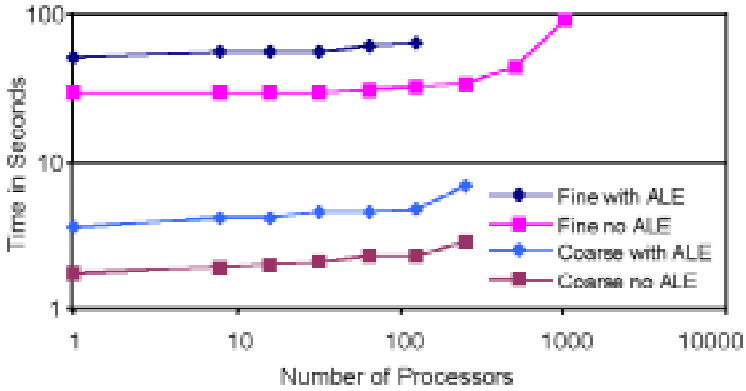

Fig. 3.6.4: Scalability of GEN1. Wall clock time per iteration on scaled problem.

Mesh motion in *Rocflo* and *Rocsolid* was tested by solving problems with an artificially enhanced burn rate. We simulated a tactical motor with a burn rate that was 1000

times the nominal value, and reduced the propellant density by the same factor so that the mass flux would be roughly the same as it is in the real rocket. This allowed us to burn away about 30% of the propellant in a few days wall clock time. The meshes maintained very high quality throughout this test.

Progress has been made on the generation of structured multiblock grids and automated production of *Rocflo* input files. Sophisticated grid models for Motor 13 and Titan IV simulations have been generated. A utility called *Makeflo* has been written (Lawlor) that can manipulate mesh files generated by the *Gridgen* commercial package. *Makeflo* can intelligently generate as many blocks as needed from a model with only a few blocks, along with the appropriate *Rocflo* mesh and connectivity files. This has reduced the turnaround time for generating new block structures and file sets for *Rocflo* from days to hours.

GEN2

An initial implementation of our GEN2 whole-system rocket simulation code was completed in September 2001. GEN2 builds on the GEN1 code, with significant improvements in modularity, accuracy, and versatility. The GEN2 architecture is shown schematically in Figure 3.6.5.

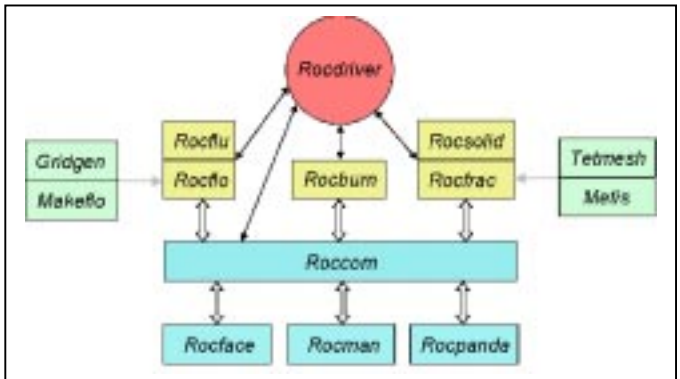

Fig. 3.6.5: GEN2 architecture.

An explicit structural dynamics solver called *Rocfrac* replaces *Rocsolid* in this first implementation of GEN2. Our short-term plan is to integrate *Rocsolid* with GEN2 so that either solver can be used for the entire domain, and our long-term plan is to enable simulations in which either solver may be active in different regions for maximum efficiency. A similar capability is planned when our explicit unstructured mesh fluids solver, *Rocflu*, is completed.

In addition to tetrahedral elements, *Rocfrac* can include cohesive volumetric finite elements to enable it to simulate crack propagation. *Rocfrac* features an ALE formulation, large deformations, and the Arruda-Boyce nonlinear constitutive model for rubbery materials such as propellant. It is MPI parallel and written in Fortran 90.

Rocface version 2.0 in GEN2 improves accuracy 20-fold (compared to the standard method from the literature in version 1.0) by solving a linear system to determine the interpolation coefficients that minimize the least squares norm of the error. Construction of an "overlay surface mesh," which is a common refinement of the meshes on either side of an interface, allows conservative interpolation in all cases, including regressing combustion interfaces with non-matching surface meshes.

Highly efficient parallel output is provided by *Rocpanda*, which is incorporated in GEN2 as a service module rather than a set of subroutine calls that must be placed in each physics module. When GEN2 is run, additional processors devoted to I/O receive messages containing output data from the processors on which the simulation runs, combine it into a manageable number of files, and write it to disk without delaying the simulation.

A new module called Roccom facilitates communication of various types. This API can handle data defined on many different kinds of meshes and provides access to service modules such as *Rocface*, *Rocpanda*, and eventually *Rocman*, which when implemented will impose interface boundary conditions between different applications to reduce the modifications required when plugging new solvers into GEN2.

Rocdriver controls the execution of the physics modules within the time stepping scheme, as well initialization, output, and stopping criteria. Each physics module may be replaced at run time by a dummy routine to allow testing the remaining modules in "stand-alone" mode. We have implemented in *Rocdriver* both the implicit/explicit (predictor-corrector) temporal coupling scheme described above and a new explicit/explicit scheme described below.

In our explicit/explicit time stepping scheme, the user chooses a system time step and the physics modules will take as many time steps whose size is limited by the Courant condition for that module to span one system time step. One option in *Rocdriver* is to poll the modules for the smallest Courant limit and set the system time step to that value. In some cases this could be prohibitively expensive, so we optionally allow each module to subcycle, with a possible reduction in accuracy due to the fact that each module receives updated interface values from the other modules only once per system time step.

In GEN2, the structures solver first takes one or more steps to reach the system time step, using values for the surface tractions and the burn rate passed to it from the fluids code on the previous step. It then passes updated interface incremental displacements and velocities to the fluids code. Next, the fluids code takes one or more steps to reach the system time step and passes the new surface tractions and burn rate to the solids code. No corrector cycles are performed in this scheme, which saves a factor of several in wall clock time per system time step and makes the speed of the GEN2 code comparable to or better than that of the GEN1 code in typical applications.

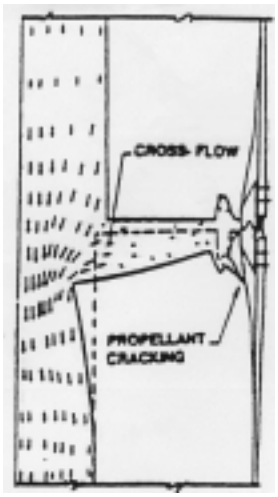

Fig. 3.6.6: Schematic drawing of Titan IV slumping failure process used in GEN2 validation.

We have performed with GEN2 a scalability test problem similar to the one described above. The timings are not directly comparable to the GEN1 results because the meshes cannot be made to match for GEN2. We find scalability superior to GEN1, with good performance above 256 processors due to our optimization of *Rocflo*. *Rocfrac* is highly scalable because it overlaps communication with computation. *Rocface* is also highly scalable, but requires a significant fraction (25%) of the wall clock time because it solves a linear system to minimize errors and because the interface is relatively large in this test problem. In more typical mesh configurations, a much smaller fraction of the faces are on an interface, and the cost of *Rocface* is proportionally lower.

We have applied GEN2 to the Titan IV SRMU propellant slumping problem (Figure 3.6.6). The domain spans only a section of the rocket near a joint slot, and a mass inflow boundary is imposed at the head end to represent the forward portion of the rocket. A subsonic/supersonic outflow condition is applied at the aft end. The cutaway view in Figure 3.6.7 shows the stress in the solid (red is low, blue is high) and cones representing the gas velocity (black is slow, magenta is medium, white is

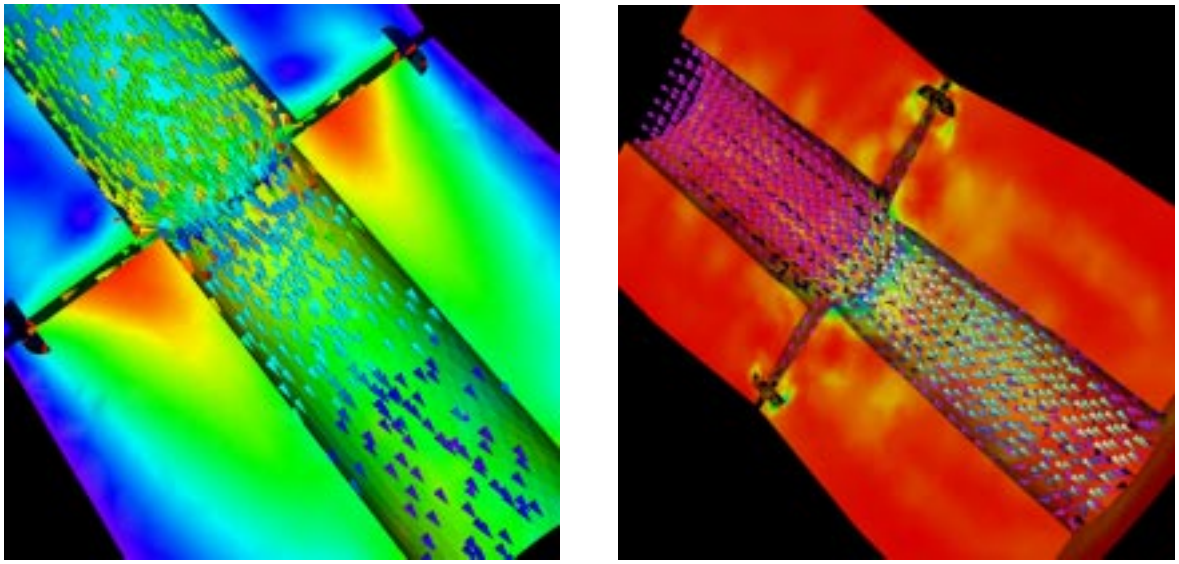

Fig. 3.6.7: Full system, 3-D simulations of Titan IV SRMU using CSAR GEN2 code. "Glyphs" depict local velocities in core flow. Color in solid propellant shows stress.

fast) after 30 ms of evolution. Gas flowing in from the left rams and flows around the inner lip of the aft segment of propellant. This results in a 3 MPa difference in surface pressure between the forward face and the region just downstream of the lip, which pulls the lip toward the center of the core flow region. The narrowing of the bore by the protruding lip raises the pressure upstream, which in turn causes even more deformation. Eventually the pressure upstream will become high enough to burst the case.

Software Engineering (Brandyberry)

Inroads have been made in formalizing our software development process. A formal documentation specification has been written, and documentation has been drafted for most current codes. Use of version control software (CVS) has become routine, and the CVS repository has been reorganized to facilitate build management for the GEN2 code. Portions of the code base have received technical peer review. A draft Verification and Validation plan has been written, and a suite of basic test problems has been assembled for the integrated code.

Visualization (Fiedler and Norris)

Rocketeer, a 3-D scientific visualization tool developed at CSAR, has continued to advance in power and sophistication. Existing visualization techniques have been expanded, new techniques have been added, and a number of significant optimizations have been applied. The "glyphs" technique has been extended for visualizing vector fields (Figure 3.6.7). A new "mesh" technique allows the user to draw all or part of a 3-D mesh. In addition, support for 2-D surface meshes (2-D elements in 3-D space) has been enhanced. These advances are also present in *Voyager*, a new MPI parallel command-line visualization tool that creates images from a series of output dumps concurrently. Figure 3.6.8 shows the scalability of *Voyager* on a Linux cluster when processing a fixed number of snapshots per processor for two cases; one in which the data files are on a shared file system, and one in which the data files are on the local disk of each node. When the shared file system is used, scalability is limited by the system's I/O bandwidth.

A full-featured implementation of the client/server version of *Rocketeer*, called *Apollo/Houston*, is now complete. *Houston*, an MPI parallel server that runs on a Linux cluster, handles reading the data and the generation of isosurfaces, glyphs, etc., speeding up the interactive generation of individual images by processing concurrently partitions of the domain decomposition in the simulation. The resulting polygonal data is sent over the network to *Apollo*, which renders the final, combined image at the desktop, exploiting hardware graphics acceleration. Development of these products will continue during the coming year. Support for large data sets will be improved, and the ability to read additional data file formats will be added by allowing the user to write plug-ins. An HDF5 plug-in will be developed first, since it should be more efficient than HDF4.

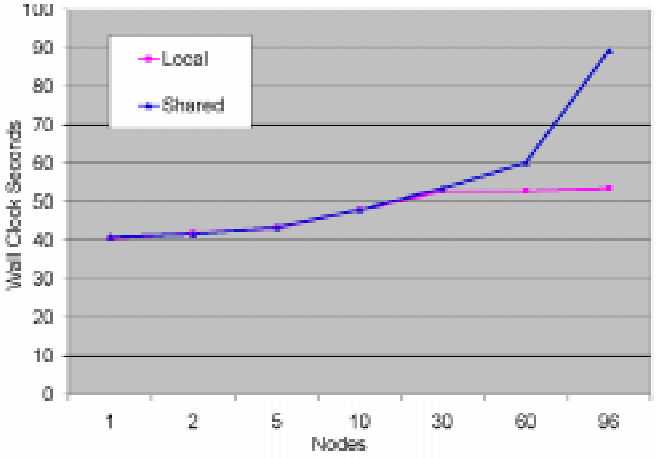

Fig. 3.6.8: Scalability of *Rocketeer*