Eleventh International Conference on Computational Fluid Dynamics (ICCFD11), Maui, Hawaii, USA, July 11-15, 2022 ICCFD11-2022-0703

An Infrastructure for Algorithmic Flexibility in Multi-fidelity and Multi-disciplinary CFD Simulations

S. Morton[*], D. McDaniel[*] and N. Hariharan[*]
Corresponding author: scott.a.morton@usace.army.mil

[*] US DoD High Performance Computing Modernization Program, USA.

Abstract: An infrastructure that is the core of a multi-disciplinary, multi-fidelity, simulation tool in the US Department of Defense High Performance Computing Modernization Program Computational Research and Engineering Acquisition Tools and Environments (HPCMP CREATE^{TM}) program that couples aerodynamics, thermochemistry, stability and control, structures, propulsion, and store separation for a large range of freestream operating conditions, Kestrel, is described. The infrastructure enables a robust and accurate capability targeting fixed-wing aircraft and is being used extensively in government and industry organizations within the US DoD acquisition community. This paper details the new flexible physics coupling strategy in the context of a notional hypersonic trajectory simulation. This strategy provides ultimate flexibility to define exactly when and how different high-fidelity physics solvers are employed during the simulation. Results are presented for the notional trajectory analysis.

Keywords: Numerical Algorithms, Computational Fluid Dynamics, Multi-fidelity Solvers, Multi-physics Simulations.

1 Introduction

Over the last several decades the US government, industry, and academia have created a rich foundation of physics-based simulation tools, including computational science and engineering (CSE) tools for fluid dynamics and propulsion (CFD), structural mechanics (CSM), structural dynamics (CSD), and electromagnetics (CEM), to name just a few. These tools have made great progress becoming applicable to the entire envelope of operation of targeted vehicles. However, over the last few decades of development the focus of these CSE tools has been on individual improvements to each of them, rather than multi-disciplinary integration of the tools into a system-level view of the vehicle.

For example, the Stability & Control engineer needs to include the effects of other disciplines, such as aeroelasticity and/or thermal elasticity, in the plant of their control system to get an accurate response during operational use in many cases. This same reasoning applies to the Loads engineer using the CFD code. It is of critical importance that the focus move from single discipline use of CFD to an integrated multi-disciplinary use at the speed necessary to impact the other disciplines, in other words at the "speed-of-relevance."

One of the driving reasons to make these changes to our focus is the need to reduce the overall timeline of the vehicle development cycle. Due to increased complexity in the system and late detection of defects, among others, the development cycle is growing from years to decades in our major aircraft programs. One sure way to reduce the timeline is to move the use of high-fidelity tools to the left on the timeline to eliminate defects in designs long before the first aircraft is manufactured or even before the first wind tunnel model is tested. The earlier a design decision is "locked-down" the more impact it has on the lifecycle cost of that system. A better understanding of the system response and performance earlier can aid in increased iterations of the design and an optimization of the vehicle can be achieved, ensuring these "locked-down" decisions are the correct ones.

In response to this need to get a system level view of the vehicle, it is critical to have an infrastructure that can be flexible enough to include other disciplines such as Propulsion, Stability and Control, Structures, and Thermal Sciences with Computational Fluid Dynamics simulations. In addition to multi-disciplines, it is also key to allow multi-fidelity of the integrated disciplines to allow long trajectory times. Probably the most noteworthy feature addition during the last year to the US DoD HPCMP CREATE™ fixed wing virtual aircraft product Kestrel[1,2] was the substantial set of changes to the workflow surrounding the coupling of different CFD-related physics capabilities in the simulation software.

Two main requirements served as the driving motivation for the changes to the infrastructure. First, the introduction of conjugate aero heating capabilities exposed limitations with the existing time-accurate coupled analysis approach for long duration trajectories of flight vehicles. The time scales of these simulations relative to the required solution time made the computations intractable. While still desiring to support high-fidelity, time-accurate simulations of smaller mission segments, it was necessary to support a more flexible coupling strategy between physics modules addressing both the frequency at which different solvers executed as well as the time integration scheme used to converge the solution (e.g. time-accurate versus quasi-steady). The second motivating factor was the growing number of users requesting the ability to run simulations with minimal or no aerodynamic modeling included (e.g. a structural heating problem with a prescribed heat flux on the aero-structural boundary). In an effort to support these and other related use cases in a general manner, some key paradigm changes were made to the standard simulation workflow. Targets of opportunity were seized to remove existing awkward interactions and workflows where possible. It is important to emphasize that all existing simulations are automatically converted to the new paradigm. A detailed discussion of the infrastructure will be provided below.

2 Infrastructure Description

From the beginning, the Kestrel development team recognized that in order to persist as a viable and effective tool, it was of paramount importance that the overarching design avoid the typical, monolithic design of so many of the simulation tools that have come before it. The natural inertia associated with the development of multi-physics software tools wants to tightly couple the various elements of the program since that is the most direct way to coordinate the various moving parts of the simulation (i.e. data shapes, execution flow, routine interfaces, etc.). However, this is realistically feasible only for single-physics simulations with only one or two developers. Even then, serious problems can arise if appropriate testing is not in place. Considering the added complexities of the multi-body, multi-physics simulations targeted by the Kestrel product and the larger number of domain experts involved, this approach is destined to produce a frustrating experience for users and the inability to make meaningful code changes as requirements and/or technologies change. Previous projects similar to Kestrel have effectively become petrified and therefore unusable for modern problems due to not being able to insert new techniques and/or features into the code with any reliable assurance of success. Indeed, during the initial releases of Kestrel when the level of complexity was relatively low, the development team sometimes succumbed to monolithic temptations in the interest of time only to have to "pay" for those decisions as improvements to support more complex use cases were added.

The Kestrel development team has accepted as fact that the only persistent requirement is that things will change in the future, whether it is requirements for new capabilities or a better algorithm or technique to utilize in an existing feature. Therefore, the overall Kestrel design is highly modular by nature, providing a way for the developers to surgically insert changes into Kestrel without having to know "everything" about all of the various components and modules. This elemental approach has analogous benefits to object oriented software design where objects can be instantiated and used without being aware of the internal implementation of the methods. The success of this approach is highly dependent on good test coverage at all levels of integration. Other projects have addressed these issues with a loosely coupled architecture that allows for a "plug-and-play" capability. However, these designs often lead to an incoherent and non-intuitive experience for users, especially for highly complex simulations such as the ones addressed here where multiple bodies are moving and/or deforming in time. In the following sections, the key tenets of the Kestrel architecture aimed at realizing the desired flexibility and adaptability of the simulation tool are discussed as well as the approach to maintain an intuitive and simplistic user experience and bring the capabilities as close to the discipline-specific user as possible. Figure 1 depicts the key elements of Kestrel and their relationships that are discussed in some detail in the following sections.

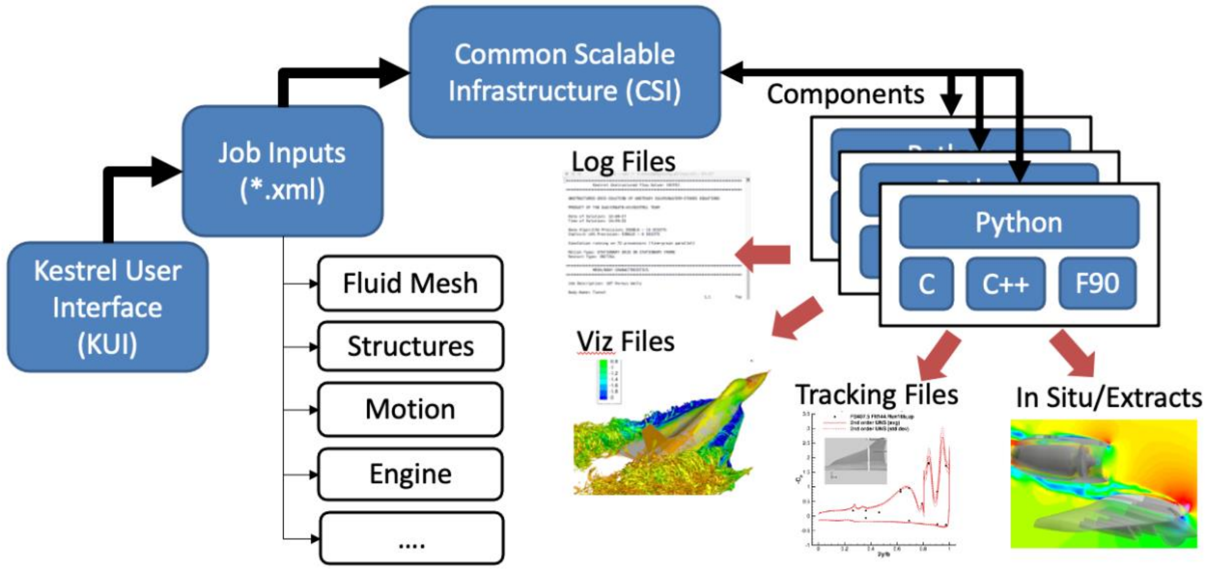

Figure 1. Main elements of the CREATE-AV Kestrel simulation tool.

2.1 Input Management and Job Setup

For complex simulations, it is very difficult for a user to represent the desired problem setup to the simulation tool in a straightforward manner. It typically requires multiple attempts (and many wasted CPU hours) before finally getting the setup correct. The overarching focus of the Kestrel job setup process is threefold: 1) the user should specify each of their input quantities only once, 2) do not require the user to convert their inputs into unfamiliar units, coordinate systems, etc. – meet them as closely as possible to their native data representation, and 3) provide mechanisms to validate the job setup as early as possible in the process. Unfortunately, this approach is not necessarily in sync with a consolidated, concise software implementation. The following sections discuss Kestrel's approach to each of these areas.

The Kestrel User Interface (KUI) serves as the one-stop shop for specifying all inputs for a simulation (see Figure 2). It attempts to provide an object-based environment for the user to relay the desired problem setup and parameter values in as natural a manner as possible while preventing the user from having to input the same information multiple times. When possible, choice lists are utilized over raw string or numeric inputs. Many numeric inputs include an automatic unit conversion capability, and it is even possible to express inputs as an algebraic expression. Support models such as a standard atmospheric model are included where appropriate in order to abstract the particular input requirements of the Kestrel physics code from the data the user has available. For example, though the Kestrel flow solver component (KCFD) desires the flow condition to be specified by Mach number, static pressure, and static temperature, Kestrel allows the condition to be specified in any number of ways to include Mach/Altitude, Reynolds/Mach/Temperature, or Mach/Density/Dynamic pressure (for aeroelastic cases).

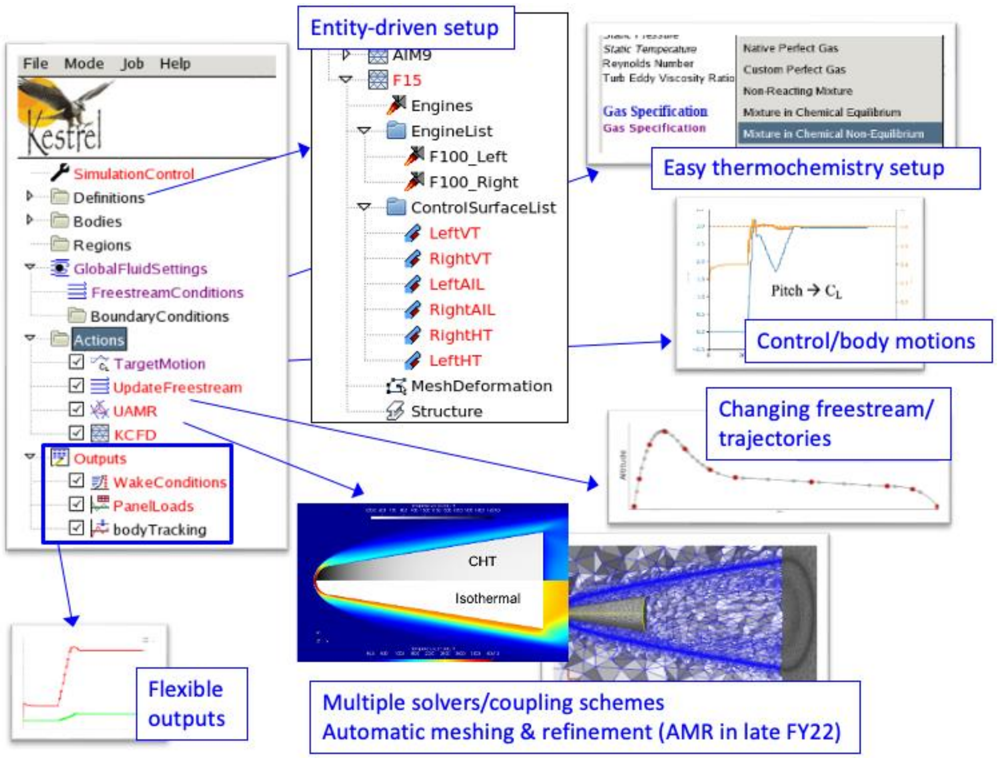

Figure 2. The Kestrel User Interface (KUI) aims to make the typical problem setup simple while keeping the hard problem setup manageable.

Inputs are grouped by functionality according to an object-based hierarchy. This approach was inspired by the techniques discussed in [4] and represents an intentional decoupling of the problem definition from the underlying component code input requirements. The main input entities are simulation control, reference conditions, body definitions, body relationships, motions, and output control. The simulation control and reference condition inputs consolidate the typical simulation parameters such as number of iterations, job names, time step size, flow conditions, etc. The body definitions allow the user to specify the characteristics of each particular type of body in the simulation. For example, a store separation simulation could have a single aircraft body definition and a single store definition even though the actual simulation could include multiple instances of each of those body types. The body definition inputs include the fluid mesh definition, boundary conditions, engines, control surfaces, and mass properties among other things. The body relationship inputs include the specification of the actual hierarchy of bodies in the simulation and their relative positioning information. Each body in the section represents an instance of a body definition. Starting with a root body positioned relative to the inertial reference frame, bodies may be added as peers (additional root bodies) or children of the body. Then, peers or children of that body may be added, and so forth.

Next, a variety of actions may be added to the simulation to incorporate dynamic effects into the simulation. Action types include mechanical and thermal structural responses, engine transients, control surface motions, prescribed rigid-body motions, and six degree of freedom (6DOF) rigid-body motion. Each action type is typically applied to one of the bodies defined in the relationship data. Extensive input validation automatically checks for invalid scenarios (e.g. a prescribed and a 6DOF motion applied to the same body at the same time). Finally, the type and amount of output data produced by the simulation may be controlled by a set of global output control parameter inputs, or per-body output specifications may be introduced for more specific requirements. In more recent years, the concept of an “action” has been expanded to include other operations that don’t necessarily represent behavior of the bodies in a simulation but rather operations to govern the behavior of the simulation. Examples include convergence detection, freestream flow condition changes, and automatic force and moment targeting.

In the end, the KUI creates an XML job input file for the simulation based on all of the various inputs provided by the user. The XML file includes all of the information necessary to define the body hierarchy, locations of input files, input coordinate systems and units, assembly instructions, and desired output coordinate systems and units, as well as all inputs needed by components of the simulation to successfully run. One of the more important behind-the-scenes pieces of the Kestrel code base is the shared utility code that collects and “normalizes” all of the disparate user inputs collected into the hierarchical XML input file into a format that each of the execution components can easily utilize. It converts all input data provided in a user-specified unit and/or coordinate system and a user-specified scale to an internal system so that individual components do not have to worry about those types of conversions.

2.2 CSI and Event-driven Architecture

The core of the modular architecture of Kestrel is the Common Scalable Infrastructure (CSI). The infrastructure is the machinery that brokers the execution flow between the various components in a simulation and handles operations that are common to each of the components. CSI is at the heart of the efforts in Kestrel to maintain flexibility and abstraction in the program design and to insulate against the inevitable changes coming in the future. There is a common misconception that CSI is the “smartest” part of the Kestrel framework. In actuality it is, by design, the most clueless part of the framework. It knows nothing and merely reacts to what is provided to it. CSI is the manifestation of the overall design choice to push use-case-specific functionality into individual components and maintain a homogenous behavior in the infrastructure. This approach naturally preserves flexibility in the Kestrel implementation and allows new features and/or new implementations of existing features to be made without disrupting the infrastructure or any of the unrelated components within Kestrel.

Each component in Kestrel includes a component application program interface (API) written in Python. These APIs implement a component-specific class based on a generic component class and contain standard methods for each component such that all components look identical to each other from the viewpoint of CSI. These standard methods include instantiation, setup, and event handling. At run time, CSI is launched in parallel on all compute ranks and utilizes the body hierarchy as discussed previously to load the XML input file for the job, convert the inputs into a common unit system, scale, and coordinate system, and then make them available to the components in a way that is easily digested. Each component specified in the XML file is instantiated in order of appearance, which includes the setup of local MPI information and the registration of events the component will respond to. Next, CSI calls a setup method for each component, which gives each component a chance to perform such tasks as extracting its desired inputs from the body hierarchy reference provided during instantiation and validating them. Finally, CSI "seeds" the event queue with the Initialize event, and that event is provided to each component for processing if it is requested. The handling of this event is where most components will perform all tasks necessary to prepare for the main simulation iterations (e.g. reading additional input files, initializing various data, etc.). At this point the individual Kestrel components take over and continue the simulation in the manner dictated by the processing of events and the publishing of new events. CSI merely manages the process of providing the events returned to it to all components that asked to handle it. This is what is meant by a "component-driven architecture". The individual components containing the "smarts" of Kestrel make the decisions on which direction the execution path should take based on the user inputs and the component algorithms.

While it is possible for a valid component to only contain the Python component API, it is common for a component to also include "lower" code levels for faster execution. All efforts are made to contain the core command and control operations at the Python level and only drop to the lower code levels as needed for efficiency. However, this is a fuzzy line since the capabilities of compiled languages have improved so much in recent years, and there is no real requirement here other than the basic desire to always use the right tool for the job. What is more important is the natural temptation discussed earlier to build "mega-components" that encapsulate a lot of different functionality. The component is the core, testable executable unit in Kestrel. Efforts are made to keep the Kestrel component landscape as elemental as possible. The more granular and focused a component is, the easier it is to maintain and test, and the more flexibility the overall simulation has with regards to execution flow. Also, more granular components imply a more highly-factored code base and less duplicate code.

There are no restrictions by CSI on the particular events that it can process. In fact, events all look identical from CSI's viewpoint. However, there is an obvious understanding that multiple components must agree to work with a common set of events in order to accomplish useful work. In effect, the manner in which events are processed and published is the "language" with which different components in a simulation communicate. The shared data items that each component works on may be considered to be the "words" used in the communication.

It is difficult to relay through words how important the event-driven architecture is to the modularity and flexibility of Kestrel. As an example, one of the earliest outgrowths of the event-driven paradigm was the ability to conduct what is referred to as a "preflight" simulation. It is possible to effectively remove the flow solver component (KCFD) from the simulation, resulting in a simulation where all other activities such as mesh surface deformation, rigid body motion, and external force application operate as in a normal simulation but the time-consuming flow solution is not executed. This allows a user to quickly execute the planned simulation in a matter of minutes to confirm the resulting body motion is as expected – another example of allowing a user to validate their simulation inputs prior to consuming precious CPU hours. The event-based approach quickly gives rise to the concept of letting the simulation determine on its own when it is ready to proceed on to the next phase of the simulation. Imagine a movie-like script that says something like "iterate on the flow solution until it is converged at which point release the store". A demand-based simulation flow like this could be invaluable for minimizing the time required to complete a complex simulation. Finally, the integration of 3rd party components from collaborators to extend the native capabilities of Kestrel is currently included through a **Error! Reference source not found.**. The event-driven framework and modular architecture of Kestrel makes this type of integration almost trivial and allows collaborators to immediately take advantage of the native capabilities within Kestrel (e.g. visualization output) that would otherwise distract from the real collaboration effort.

A Data Warehouse (DW) facilitates the sharing of data between components on the same compute rank. The DW provides mechanisms for a component wrapper to register data items that it produces with the DW and places the item in the warehouse in a particular format (language). Then, any component can safely check to see if a data item exists in the warehouse, and if so, request the data item in any language. As long as the proper conversion routines have been supplied by the registering component (or the DW itself), the DW will automatically work with the owning component to provide a reference to the desired data. The heavyweight data are still passed between components by reference such that when one component modifies a common data array the change is immediately represented in any other component referencing the data.

2.3 Multi-Disciplinary Components

The components that plug into the CSI provide the needed physics for the problem at hand and can also interact with other components through the CSI to provide the multi-disciplinary simulations desired by the User. The Components subscribe to the events discussed above (e.g. iterate, sub-iterate, create restart) and also produce or subscribe to data elements in the Data Warehouse. Figure 2 below depicts the various finite volume and finite element fluid dynamics solvers (e.g. KCFD, COFFE, SAMAir) and their interactions with various modal and finite element structures solvers (e.g. Modal/SD, Sierra/SD) and a conjugate gradient heat transfer solver (e.g. Sierra/TF Aria). The interaction components described in Figure 2 are PUNDIT, allowing multiple fluid solvers to transfer data using interpolation of mesh points, and FSI, allowing fluid solvers to transfer data to/from structural solvers through interpolation of their mesh points. Both PUNDIT and FSI are components that are completely automated with no interactions by the User necessary.

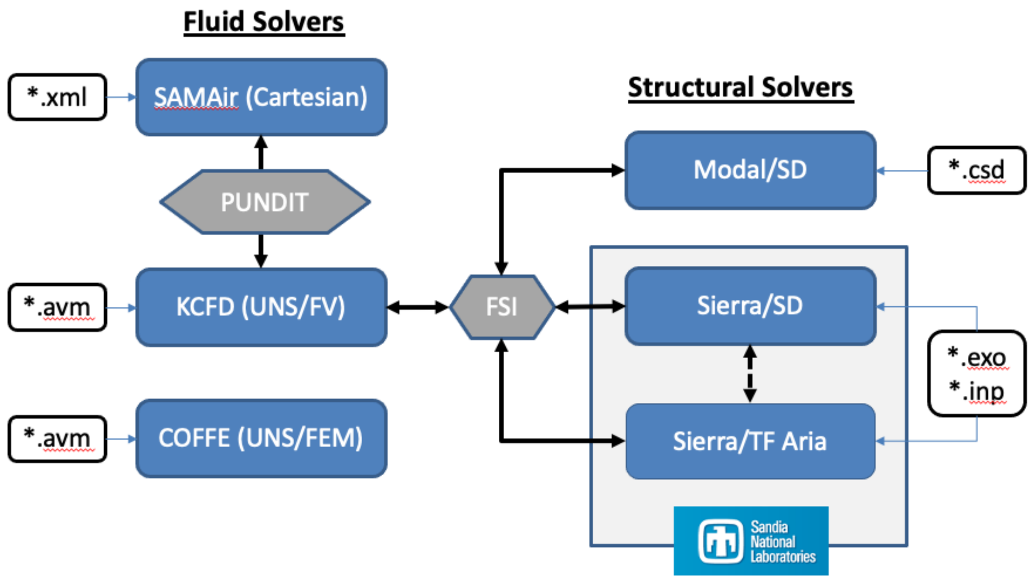

Figure 3. Schematic showing fluid and structural solvers currently present in CREATE-AV Kestrel.

2.3 Multi-Fidelity Multi-Physics Coupling

One of the most important and sweeping changes to Kestrel over the last several years is a result of requirements from the hypersonic vehicle design community. Hypersonic flight requires additional physics to be included in the simulation with varying time scales (e.g. thermal-chemistry, ablation, fluid-thermal-structural interactions). It would no longer be possible to simply run time accurate at the smallest relevant scale (typically CFD with on the order of 10^{-5} sec time step) and have reasonable simulation times. It was also necessary to run for long simulation times of 10's minutes in order to account for thermal soaking of the vehicle in hypersonic flight. These two requirements led to infrastructure changes that allow the disparate solvers to execute at different intervals in the simulation.

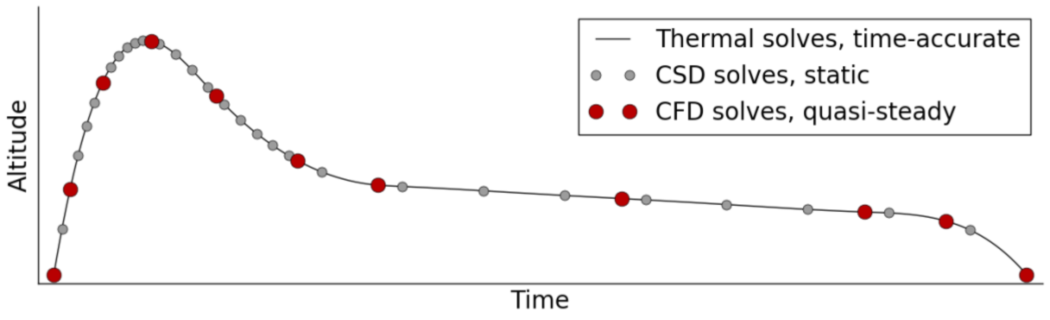

Figure 4. Notional vehicle trajectory with non-uniform physics coupling and time solution schemes.

Figure 4 represents a notional example of an aeroheating simulation during a hypersonic trajectory that might require the thermal solver to execute every time step while the structural solver and CFD solver execute in a quasi-steady mode at orders of magnitude larger time steps, making the overall simulation of 10's of minutes affordable. The simulation begins by assuming the flow field is steady and converging the flow field around the initial thermal state. The flow field is then held fixed while the thermal solver marches time forward at a time step appropriate for the thermal domain. At user-specified times, the steady flow field is recomputed around the most recent thermal state, and the new surface heat flux values are used as the thermal solver continues in time. The user may elect to define convergence criteria to terminate each of the fluid solve “segments” or may just have them go for a specified number of iterations.

The simulations described above can also span hundreds of thousands of feet in altitude, causing issues with the freestream conditions relevant to the overall simulation. Kestrel now supports updating the freestream conditions during a simulation. A user may provide a file that describes how Alpha, Beta, and the simulation’s “Known Conditions” (e.g. “Altitude-Mach”) change as a function of time. Every point in the file represents a discrete jump in the freestream conditions, and Kestrel will re-initialize its flow field and reference state to the new conditions.

During the course of making these changes, another important decision was made to help with multi-fidelity simulations. In previous versions of Kestrel, CFD solvers have been the assumed drivers of the simulation and were always called during the simulation. In order to allow surrogate aero models to be incorporated into the simulation, this assumption had to be eliminated. Now a user can choose “no CFD solve” or replacement of the CFD solve with a surrogate model. This new strategy opened Kestrel to new use cases like thermal-structural only simulations with a prescribed aero input, as well as many others. The next section describes some multi-fidelity, multi-physics simulations that take advantage of the new flexibility in the Kestrel architecture.

3 Example Applications

This section presents two examples of the flexibility of the infrastructure to multi-physics and multi-fidelity solvers. The first example is an aero-heating simulation using a conjugate gradient heat transfer solver along with the CFD solver, and the second example is a long duration trajectory simulation with disparate time intervals for the thermal solver and the CFD solver.

3.1 Static Canonical Body Conjugate Gradient Heat Transfer

Thermal interactions between fluid and structural domains are a fundamental aspect of aero-structural simulations. Flight in the hypersonic regime brings about non-negligible coupling between aerodynamic pressure, shear, and thermal loads, heat transfer and structural deformation. While emphasis on the hypersonic flight regime has been the primary driver for development of this capability in Kestrel, there are also a number of other aero-heating use cases that require the capability (e.g. turbomachinery, jet impingement, store and aircraft thermal load management).

Kestrel has the ability to solve heat transfer problems by coupling with the "Aria" solver from Sierra/TF (Sandia National Labs). Aria is a parallel, multi-physics solver capable of solving a wide variety of partial differential equations, including the energy equation needed to solve heat transfer in a solid. With the introduction of Aria into Kestrel, the array of available solvers can quickly get confusing. As discussed earlier, Figure 3 shows a schematic of the different fluid and structural solvers included in Kestrel, as well as the current couplings between the solvers. The native fluid-structure interaction (FSI) component has been extended to support both mechanical and thermal information transfer.

When coupled with Aria, KCFD uses the temperature distribution on the interface surface from Aria as a localized input into the no-slip wall boundary condition. After moving the time step forward, KCFD will have computed a new heat flux on the interface, which is then passed to Aria as a specified heat flux boundary condition. The coupling is such that the solvers can exchange information at every time step, or each solver can perform an independent number of time steps in between coupling. Figure 5 shows how accounting for heat transfer in a simulation impacts the shock profile and temperature field around a blunted cone at Mach 20 due to the energy transfer into the structure.

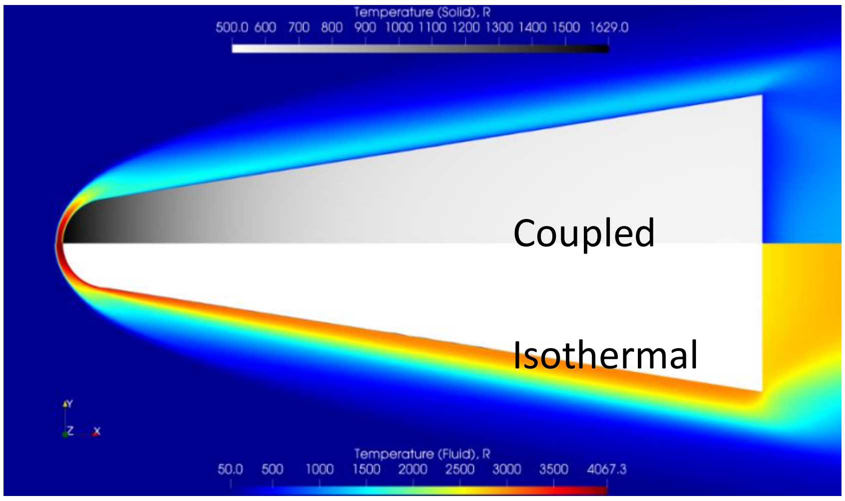

Figure 51. Sphere-nosed cone flow solution at Mach 20 with coupled aero-thermal physics (top) and a fixed isothermal solid temperate boundary condition (bottom).

3.2 Long Trajectory Multi-Physics Multi-Fidelity Simulation

This example demonstrates the ability to specify various time steps for multiple physics solvers in a simulation. Consider a hollow cylinder made of 321 Stainless Steel with the inner wall held at a constant 294.4 K. Figure 6 depicts the quad-dominant aerodynamic solver's mesh and the structured hollow cylinder mesh for the thermal solver. The cylinder is "flown" through a fictitious trajectory climbing from sea level to 15 km while accelerating from Mach 1.5 to 6.5 over a time period of 5 minutes at which point it holds those conditions constant for a period of 5 minutes before returning to sea level and Mach 1.5 conditions (see Figure 7). The simulation is performed in air as a perfect gas and using the Menter SST turbulence model. A steady flow solution is computed with CFL=1000 (until converged or 10k sub-iterations) every 10 seconds of simulation time in response to the freestream conditions being updated and using the surface temperature from the most recent thermal solution. The thermal solution is solved time-accurate with a time step size of 0.01 sec using the most recent surface heat flux distribution from the flow solve.

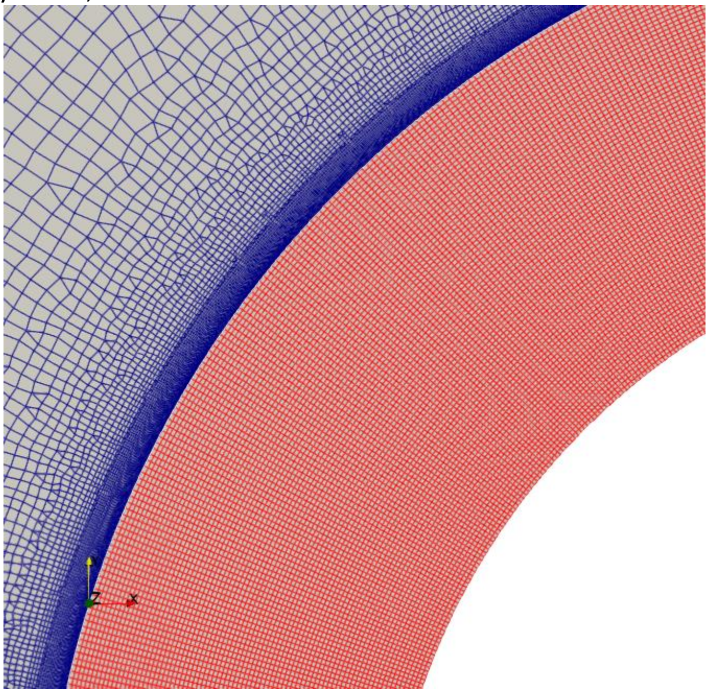

Figure 6. Close-up view of aero (fluid) quad-dominant grid (blue) with ~80k triangle cells and ~1.6M quad elements and structured grid for hollow cylinder (red) with ~77k quad elements.

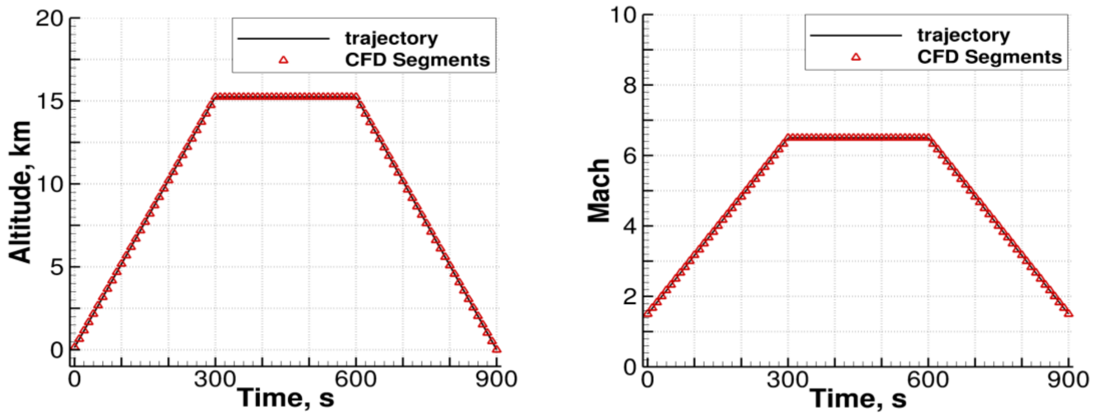

Figure 72. Altitude and Mach number time histories for notional hypersonic trajectory with symbols showing time points where a new fluid solution is computed in a quasi-steady coupling methodology.

Figure 8 (left) shows the resulting surface maximum temperature as a function of time for the simulation. The maximum temperature increases fairly linearly during the climb to a 15 km altitude and an increase in velocity to the maximum Mach number of 6.5 with a maximum temperature close to 1200 K that then falls off fairly linearly during the descent. Figure 8 (right) shows the distributed surface temperatures at 6 time slices in the trajectory.

Figure 9 shows temperature contours of both the fluid and structural domains at six times during the trajectory (150 sec, 300 sec, 450 sec, 600 sec, 750 sec, and 900 sec). A few of the features worth noting in the contours are the position of the bow shock, high heating in the stagnation region and the extent of the high temperature region around the cylinder at different points in the flight trajectory.

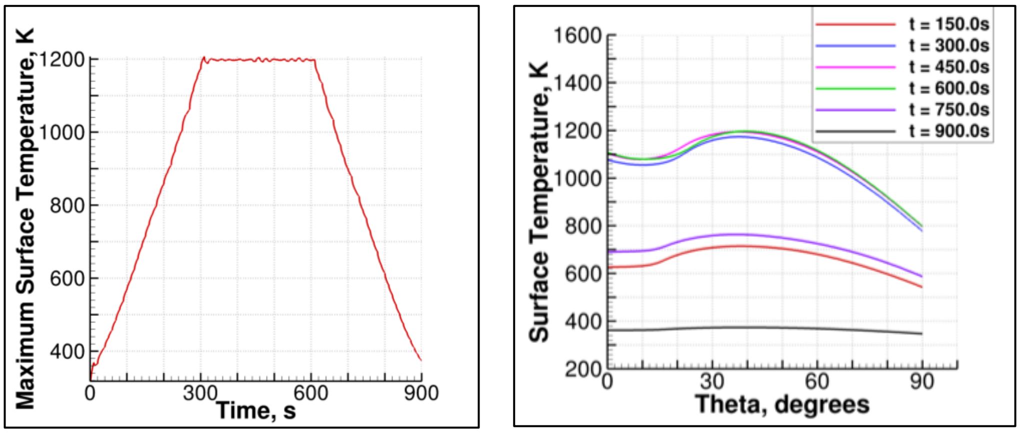

Figure 8. Maximum surface temperature as a function of time (left) and surface distribution at selected time points (right) for the aeroheated cylinder simulation.

Table 1 below shows the dramatic variation in cost of the simulation depending on the chosen simulation strategy. The time to compute the entire trajectory for the proposed time accurate thermal solve with quasi-steady aero solve and convergence detection is 28 hours, whereas a quasi-steady aero solve with a specified number of convergence sub-iterations is 54 hours. Compare this to the traditional fully coupled time accurate method with an under resolved aero time step of 0.001 sec requiring 87.5 hours and a more representative time accurate time step of 0.00001 sec requiring 8,750 hours.

Table 1. Actual and predicted run times of hypersonic cylinder simulation (based on 44 compute cores on Onyx at the ERDC DSRC).

Present Work	**No Convergence Detection**	**Couple every iteration Δt=0.001 (predicted)**	**Couple every iteration Δt=0.00001 (predicted)**
28 hrs	54 hrs	87.5 hrs	8,750 hrs

Clearly the proposed method is more suitable for a design cycle and the accuracy can be evaluated for some subset of conditions using the fully coupled time accurate method. The flexibility in the Kestrel infrastructure and the intuitive User Interface allows users to find the correct simulation strategy for their application in a single code.

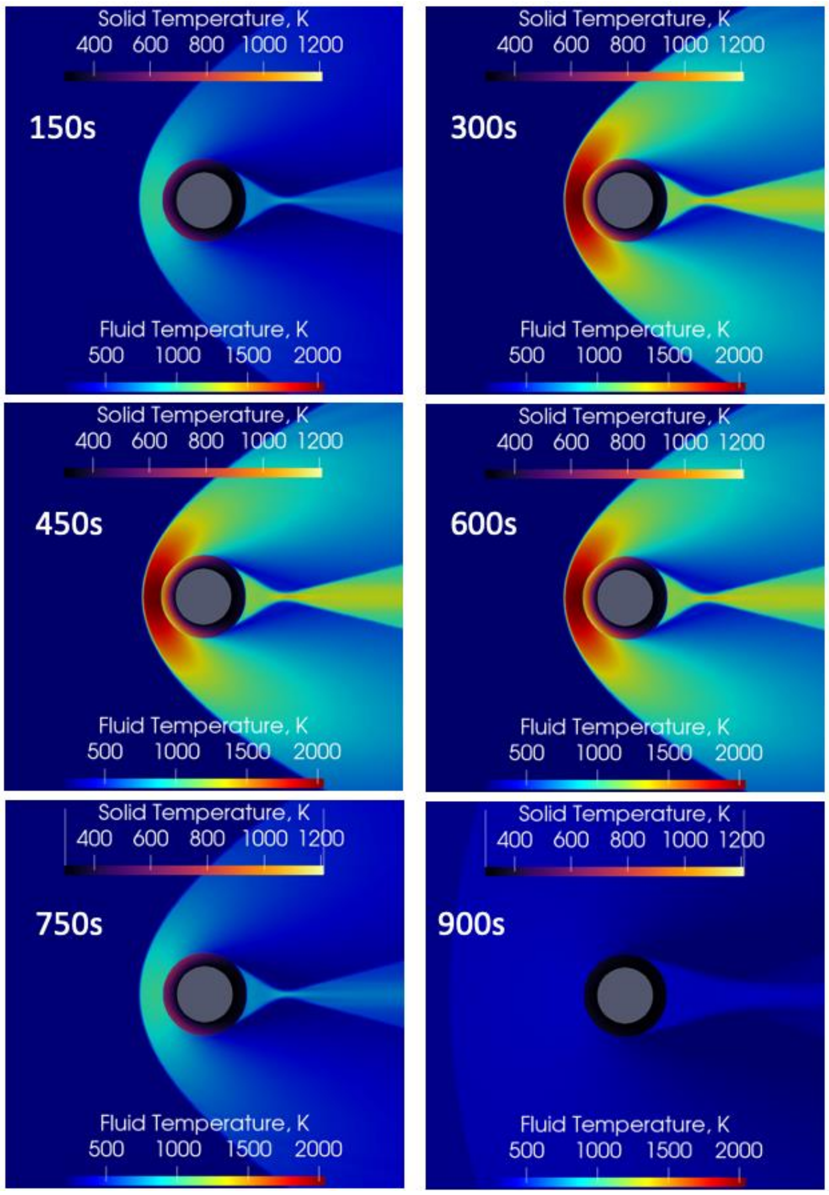

Figure 93. Fluid and structure temperature contours at selected time points for the aeroheated cylinder simulation. Note the colormaps for the fluid and structure are not equal.

4 Conclusions

Kestrel is a multidisciplinary, high-fidelity simulation tool targeting robust and accurate engineering solutions for the DoD acquisition community. Solutions for full aircraft configurations containing complex, unsteady aerodynamics, thermochemistry, mechanical and thermal fluid-structure interaction, rotating turbomachinery, control surfaces, and rigid-body motion are possible in a production environment. Intuitive and error-free job setup and validation is a priority, and a unique, event-based execution paradigm which integrates various modular physics components helps to minimize the complexity associated with a growing number of multidisciplinary simulation requirements. The recent changes to the infrastructure enabling multi-physics at multi-fidelity have been proven to be computationally efficient for long duration fluid-thermal simulations. Extending this capability to the fluid-thermal-structural interaction problem will prove extremely useful to the hypersonic system design community and is a priority for the Kestrel Development Team.

5 Acknowledgements

Material presented in this paper is a product of the HPCMP CREATE^{TM}-AV Element of the Computational Research and Engineering for Acquisition Tools and Environments (CREATE) Program sponsored by the U.S. Department of Defense HPC Modernization Program Office. We would especially like to thank all of the CREATE-AV Quality Assurance and Kestrel team members for their hard work and invaluable contributions.

References

[1] Morton, S.A., McDaniel, D.R., Sears, D.R., Tuckey, T.R., Tillman, B., “Kestrel – A Fixed Wing Virtual Aircraft Product of the CREATE^{TM} Program,” 47th AIAA Aerospace Sciences Meeting, 5 - 8 January 2009, Orlando, Florida.

[2] McDaniel, D., Tuckey, T., “Multiple Bodies, Motion, and Mash-Ups: Handling Complex Use-Cases with Kestrel,” AIAA Paper 2014-0415, 52nd Aerospace Sciences Meeting, National Harbor, Maryland, Jan. 13-17, 2014.

[3] McDaniel, D. and Tuckey, T., “HPCMP CREATE^{TM}-AV Kestrel New and Emerging Capabilities,” AIAA Paper 2020-1525, AIAA SciTech 2020 Forum, 6-10 January 2020, Orlando, FL.

[4] Murman, S.M., Chan, W.M., Aftosmis, M.J., and Meakin, R.L., “An Interface for Specifying Rigid-Body Motions for CFD Applications,” AIAA Paper 2003-1237, 41st AIAA Aerospace Sciences Meeting, 6-9 January 2003, Reno, Nevada.