A Multi-Architecture Approach for Implicit Computational Fluid Dynamics on Unstructured Grids

Gabriel Nastac[1], Aaron Walden[1], Li Wang[1], Eric Nielsen[1], Yi Liu[2], Matthew Opgenorth[3], Jason Orender[4], Mohammad Zubair[4]

[1]NASA Langley Research Center

[2]National Institute of Aerospace

[3]Sierra Space

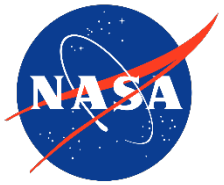
[4]Old Dominion University

Copyright 2023 United States Government, as represented by the Administrator of the National Aeronautics and Space Administration, National Institute of Aerospace, Sierra Space Corporation, and Old Dominion University. Published by the American Institute of Aeronautics and Astronautics, Inc. with permission.

Motivation

- Graphics processing units (GPUs) have over an order of magnitude greater concurrency than multicore CPUs
  - NVIDIA A100 GPU: 221,184 logical threads
  - AMD EPYC 7742 CPU: 256 logical threads
- Extracting potential hardware performance requires exposing more parallelism
- GPUs are the key technology for next-generation HPC
  - Seven of the top 10 supercomputers (based on TOP500) use GPUs
  - Half of the top 100 supercomputers in the world use GPUs
- U.S. exascale systems rely on GPU acceleration

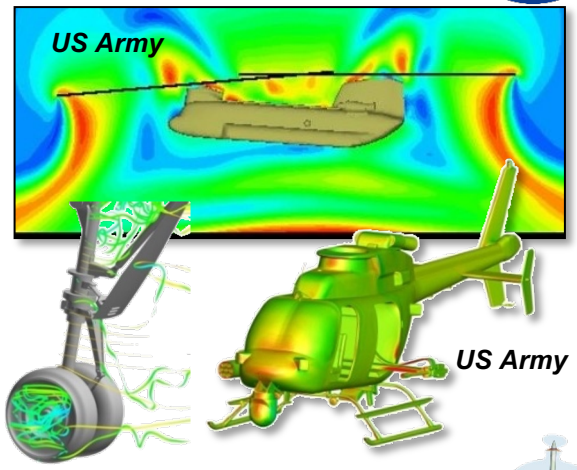
Motivation (cont.)

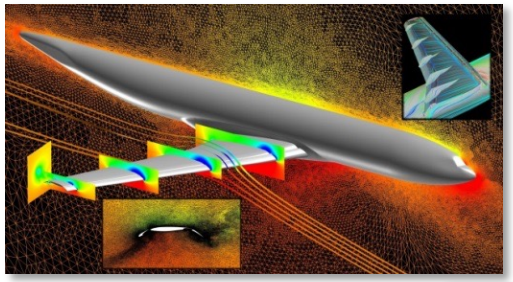


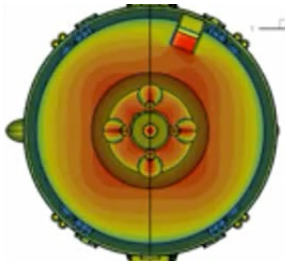
- Difficult or impossible to compile and run legacy software on GPUs efficiently without significant refactoring
- Most vendors have vendor-specific paradigms
- Many programming models exist to run on GPUs, e.g., (not exhaustive)
  - NVIDIA CUDA C++, AMD HIP, Intel DPC++, OpenCL, ISO C++, Vulkan Compute, SYCL, OpenMP, OpenACC, Kokkos, RAJA
- Ideally, one would write in a standardized specification supported by major hardware vendors and achieve satisfactory performance across contemporary HPC architectures
- **Performance is paramount for GPU adoption; it must be cost-effective and performant enough to potentially rearchitect software**
  - **Especially so as cloud computing becomes more prevalent**

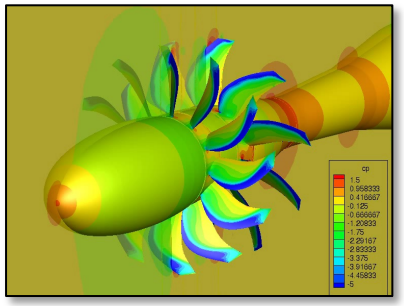
NASA Fully Unstructured Navier-Stokes 3D (FUN3D)

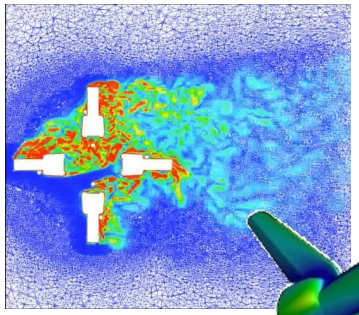
- Established as a research code in late 1980s; now supports numerous internal and external efforts across the speed range
- **Solves 3D Navier-Stokes equations using node-based finite-volume approach on mixed element unstructured grids**
- Fully **implicit formulations** are generally used to integrate the equations
- General dynamic mesh capability: any combination of rigid / overset / morphing grids, including 6-DOF effects
- Aeroelastic modeling using mode shapes, full FEM, CC, etc.
- Constrained / multipoint adjoint-based design, mesh adaptation
- Distributed development team using agile/extreme software practices including 24/7 regression, performance testing
- Capabilities fully integrated, online documentation, training videos, tutorials

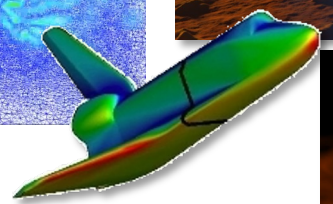


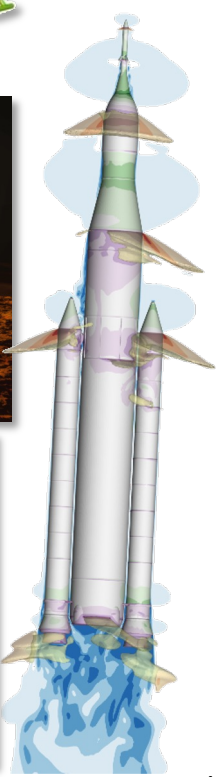


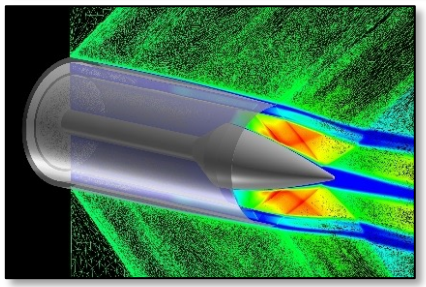


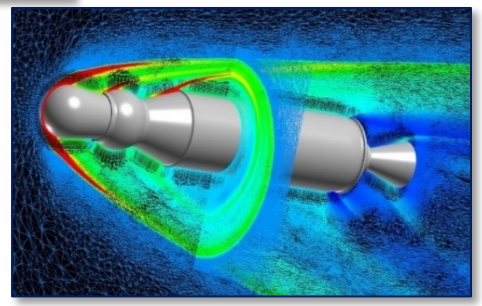


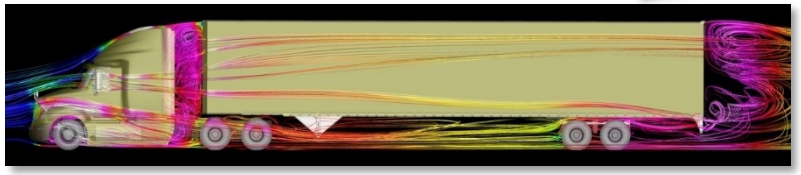




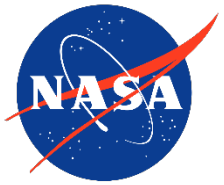








FUN3D Library for Universal Device Acceleration (FLUDA)



- **Finite-volume on unstructured grids is primarily memory-bound, so performance should scale with memory bandwidth to 1st order**
  - Limited success with early GPU efforts to achieve performance parity with highly optimized legacy Fortran
- To achieve high performance on NVIDIA GPUs, FLUDA was created in 2017 as a CUDA port of FUN3D's flow solvers
  - Identical data structures to Fortran
  - Double precision variables
- See past papers for details on GPU port and optimizations
- AMD GPU support was extended using a thin abstraction layer similar to CUDA C++ and AMD HIP
- Abstraction enables:
  - NVIDIA GPU usage through CUDA C++
  - AMD GPU usage through AMD HIP
  - Intel GPU usage through SYCL
  - CPU usage through ISO C++
- CPUs run the GPU-oriented code with a single thread
- Preprocessing macros comprise ~500 lines of code
- **Our hypothesis is that modern, superscalar CPUs will better tolerate GPU-oriented code than the reverse**

```
#include "platform.h"

#define EDGE_CONNECTIVITY(j,i) edge_connectivity[ (j) + (size_t)(i)*2 ]

__DEVICE__
double f(const double varl, const double varr)
{
  return 0.5*(varl+varr);
}

__GLOBAL__
void example_kernel(const int nedges, const int* edge_connectivity, const double* var,
    const double* area, double* residual)
{
    int n = BLOCKIDX_X * BLOCKDIM_X + THREADIDX_X;
    int grid_size = BLOCKDIM_X * GRIDDIM_X;

    // CPU_GPU(for(;n < nedges; n++), if (n < nedges))
    for(;n < nedges; n += grid_size)
    {
     int nodel = EDGE_CONNECTIVITY(0,n); // Left state
     int noder = EDGE_CONNECTIVITY(1,n); // Right state

     double flux = f(var[nodel],var[noder]) * area[n];

     ATOMICADD(&residual[nodel],  flux);
     ATOMICADD(&residual[noder], -flux);
   }
}
```

Example FLUDA edge (dual-face) kernel

Parallel Approach

- **Kernels must expose sufficient parallelism to attain high performance**
- Jacobian example
- Loop over edges/dual-faces of grid
  - Construct NxN Jacobians
  - Add to global data
- 5-species, two-temperature gas model (N = 10)
- **Hierarchical parallelism is required for optimal GPU performance**

Naïve parallel edge (dual-face) Jacobian kernel

```
template<class CONFIG>
__GLOBAL__
void example_kernel_edge_parallelism(...)
{
  constexpr int N = CONFIG::N;
  // local variable declarations

  int n = BLOCKIDX_X*BLOCKDIM_X+THREADIDX_X;
  int grid_size = BLOCKDIM_X*GRIDDIM_X;

  for(;n < nedges; n += grid_size)
  {
    // Compute derived state and store

    // Compute and store Jacobians (NxN)
    for (int k=0;k<N;k++) {
        for (int j=0;j<N;j++) {
            // Compute (j,k) components
        }
    }

    // Add Jacobians to global data
  }
}
```

Hierarchical parallel edge (dual-face) Jacobian kernel

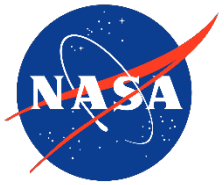
```
template<class CONFIG>
__GLOBAL__
void example_kernel_hierarchical_parallelism(...)
{
  constexpr int N = CONFIG::N;
  // local and shared variable declarations

  int n = BLOCKIDX_X*BLOCKDIM_Y+THREADIDX_Y;
  int grid_size = BLOCKDIM_Y*GRIDDIM_X;

  for(;n < nedges; n += grid_size)
  {
    if (THREADIDX_X == 0) {
      // Compute minimum derived state
      // and store in shared memory
    }

    __SYNCTHREADS();

    for (int i=THREADIDX_X;i<N*N;i+=BLOCK_DIM_X)
    {
      int j = i / N;
      int k = i % N;

      // construct (j,k) components in place
      // and add to global data
    }
  }
}
```

Architecture	Implementation	Speedup	Hardware MBW Ratio
Intel Skylake 6148 (40 cores)	Fortran	0.73	1.00
Intel Skylake 6148 (40 cores)	FLUDA Naïve	1.00	1.00
Intel Skylake 6148 (40 cores)	FLUDA Hierarchical	0.89	1.00
NVIDIA 16 GB SXM V100	FLUDA Naïve	0.58	3.52
NVIDIA 16 GB SXM V100	FLUDA Hierarchical	4.41	3.52

Hierarchical kernel 7.6x faster than naïve kernel on GPU, but only 1.1x slower on CPU

Performance scales with memory bandwidth ratio for optimal implementations

Multi-Architecture Approach



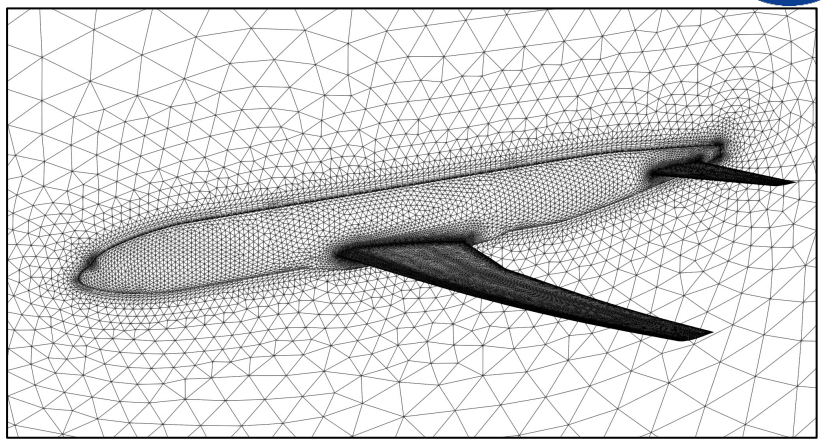
- Architecture-specific code used when employing hierarchical parallelism is much worse
- CPU and GPU code divergence is <2% of code base
- Linear solver (~50% of run time, <1% of code base) has architecture optimized implementations
- We achieve high percentage of peak memory bandwidth for linear solver (60-80% across architectures)
- Other key kernels are generally achieving 50% of peak or better according to profilers
- Performance across GPU architectures is obtained through autotuning of thread block parameters
  - Many kernels are templated on gas model (e.g., number of species) and element type
  - Each template combination has tuned threading parameters for each architecture

Results Overview

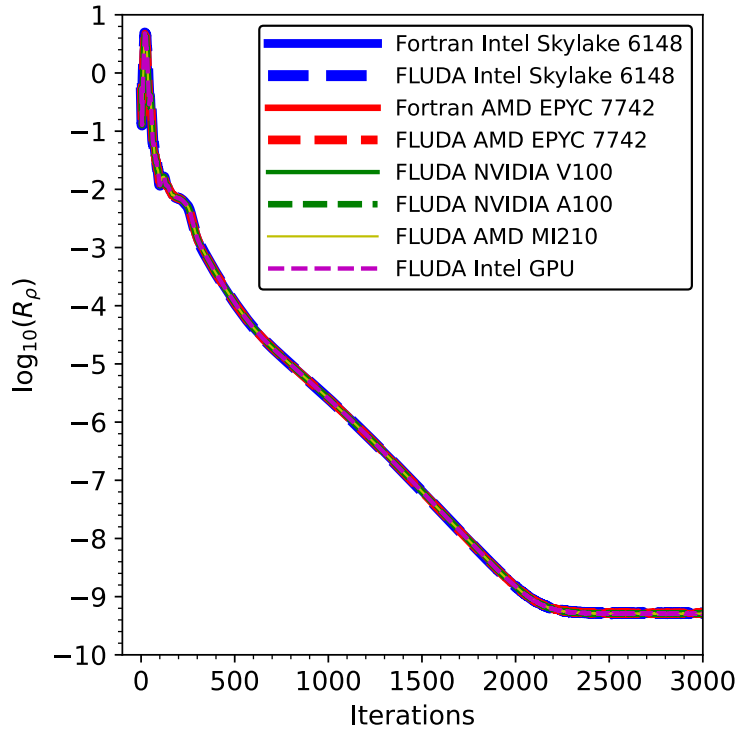
- Various results shown for a wide breadth of physics and applications
- Performance results are fastest times obtained after testing a variety of compilers and optimization flags
  - GPU-aware MPI is **not** currently used
- Results are run mainly on NAS hardware
- RANS simulation of NASA Common Research Model (CRM) at Transonic Conditions
  - Correctness
  - Device-level performance
- WMLES of High-Lift NASA CRM
  - Performance at scale and unsteady flow
- Tilt Rotor Aeroacoustics Model (TRAM)
  - Performance of moving-grid capabilities
- Hypersonic Crew Exploration Vehicle (CEV) and Sierra Space Dream Chaser© spaceplane
  - Performance of thermochemical nonequilibrium flows

RANS Simulation of the NASA CRM at Transonic Conditions

- Please see paper for details
- 3.7M points, 10M mixed elements
- SA-QCR2000
- Time to machine zero convergence
- **Hardware memory bandwidth (MBW) is theoretical and vendor reported**



Architecture	Implementation	Time [min]	Speedup	Hardware MBW Ratio	Hardware MBW [GB/s]
Intel Skylake 6148 (40 cores)	Fortran	56	0.84	1.00	256
Intel Skylake 6148 (40 cores)	FLUDA	47	1.00	1.00	256
AMD EPYC 7742 (128 cores)	Fortran	29	1.62	1.60	409.6
AMD EPYC 7742 (128 cores)	FLUDA	27	1.72	1.60	409.6
NVIDIA 16 GB SXM V100	FLUDA	12	3.91	3.52	900
NVIDIA 40 GB SXM A100	FLUDA	7	6.55	6.05	1555
AMD MI210	FLUDA	10	4.55	6.40	1638.4

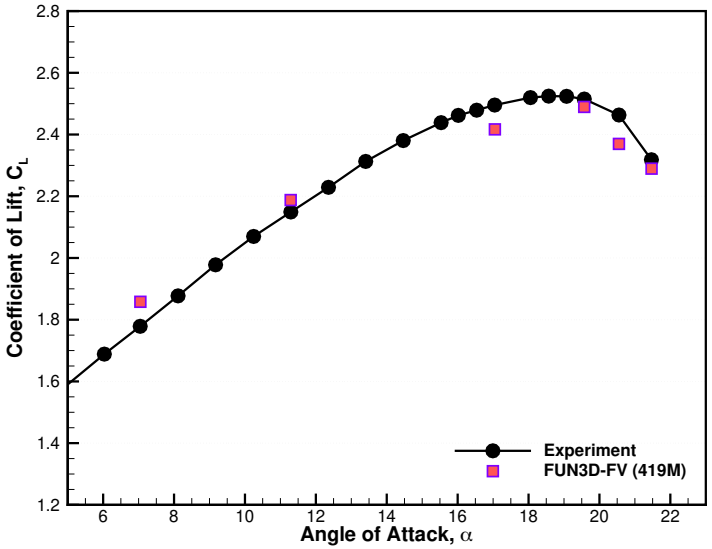


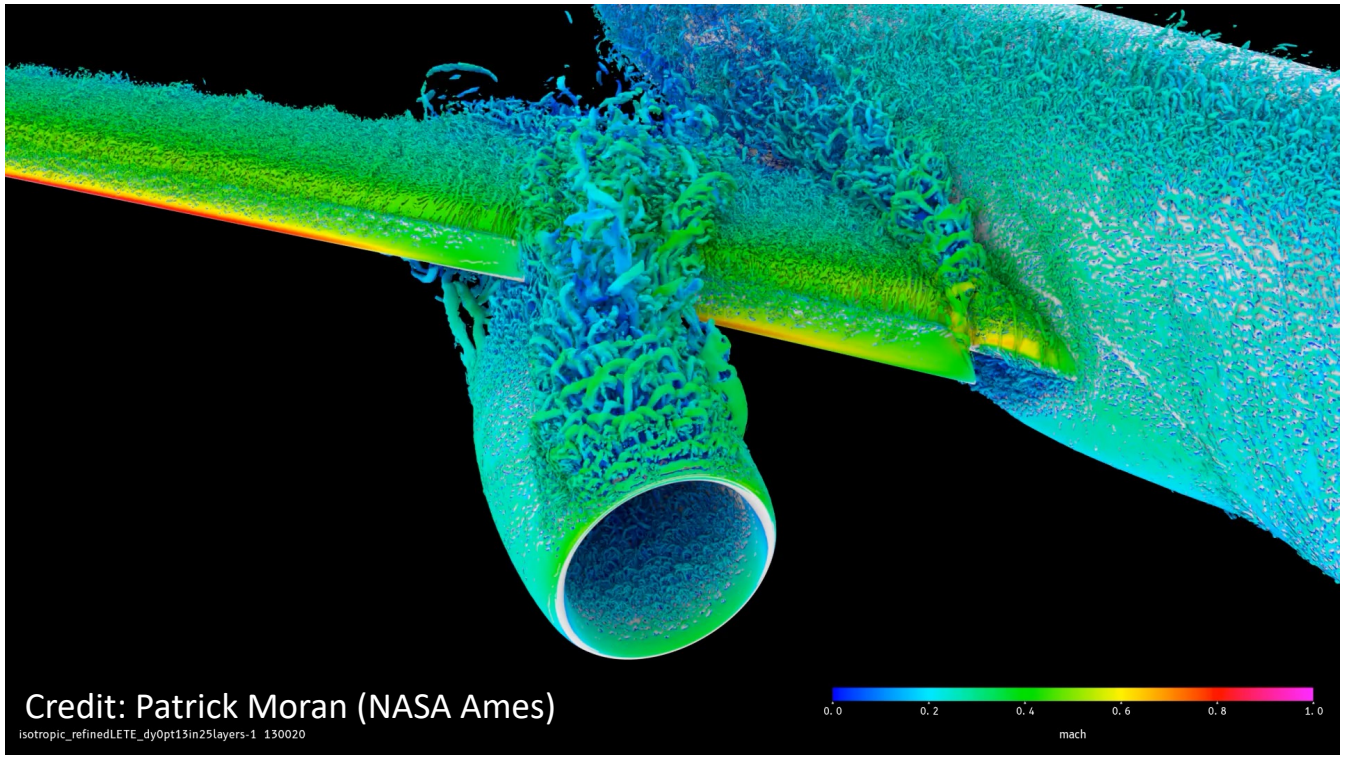
Speedup = Hardware device-level performance normalized to FLUDA CPU

Performance scales with memory bandwidth ratio

WMLES of High-Lift NASA CRM

- Please see paper for details
- 419M points, 812M mixed elements
- O(50) Convective Time Unit (CTU) for statistically stationary result
- Viscous term optimizations are expected to speedup time per CTU by 1.2x in future





Architecture	Implementation	Time per CTU [min]	Speedup	Hardware MBW Ratio
200 AMD EPYC 7742 (25,600 cores)	Fortran	81	0.87	1.00
200 AMD EPYC 7742 (25,600 cores)	FLUDA	69	1.00	1.00
108 NVIDIA V100	FLUDA	60	2.13	2.20

Speedup = Normalized hardware device-level performance
1 V100 ~ 2.13 EPYC nodes ~ 273 cores

Performance scales with memory bandwidth ratio

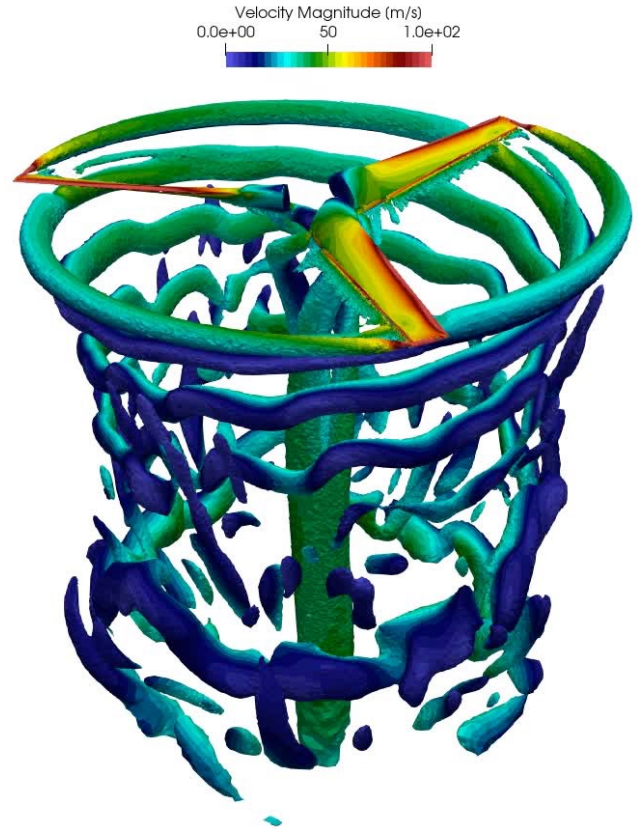
Tilt Rotor Aeroacoustics Model (TRAM)

- Please see paper for details
- 20M points, 97M mixed elements
- SA-Delayed DES
- Figure of merit (FM, hover efficiency) within 2% of experimental data
- FM converges after approximately 4 revolutions

Architecture	Implementation	Time per 4 Revolutions [min]	Speedup	Hardware MBW Ratio
20 AMD EPYC 7742 (2,560 cores)	Fortran	77	0.84	1.00
20 AMD EPYC 7742 (2,560 cores)	FLUDA	65	1.00	1.00
16 NVIDIA V100	FLUDA	34	2.38	2.20

Performance scales with memory bandwidth ratio

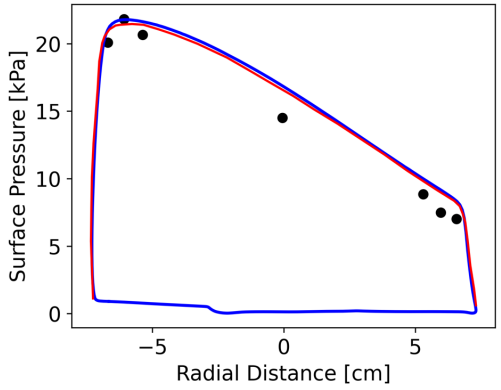
Speedup = Normalized hardware device-level performance
1 V100 ~ 2.38 EPYC nodes ~ 305 cores

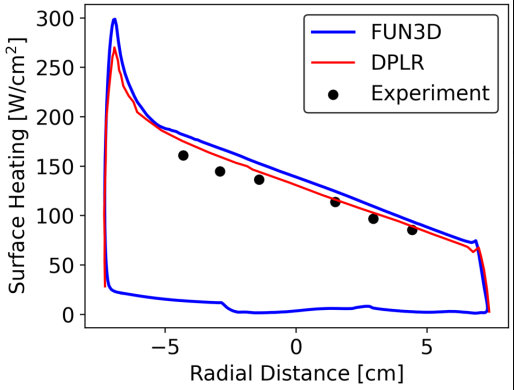


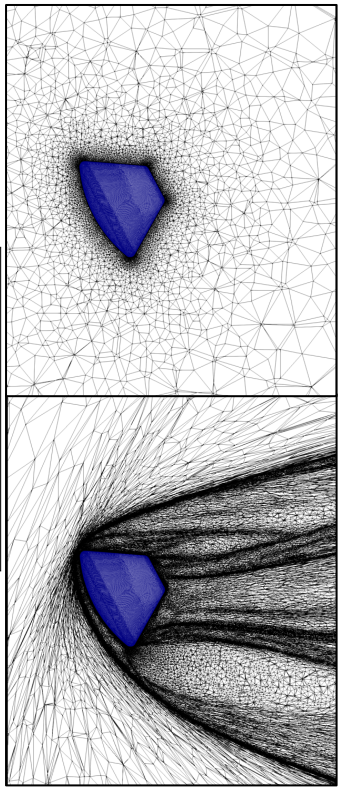
Isosurface of Q-criterion (0.001 1/s)

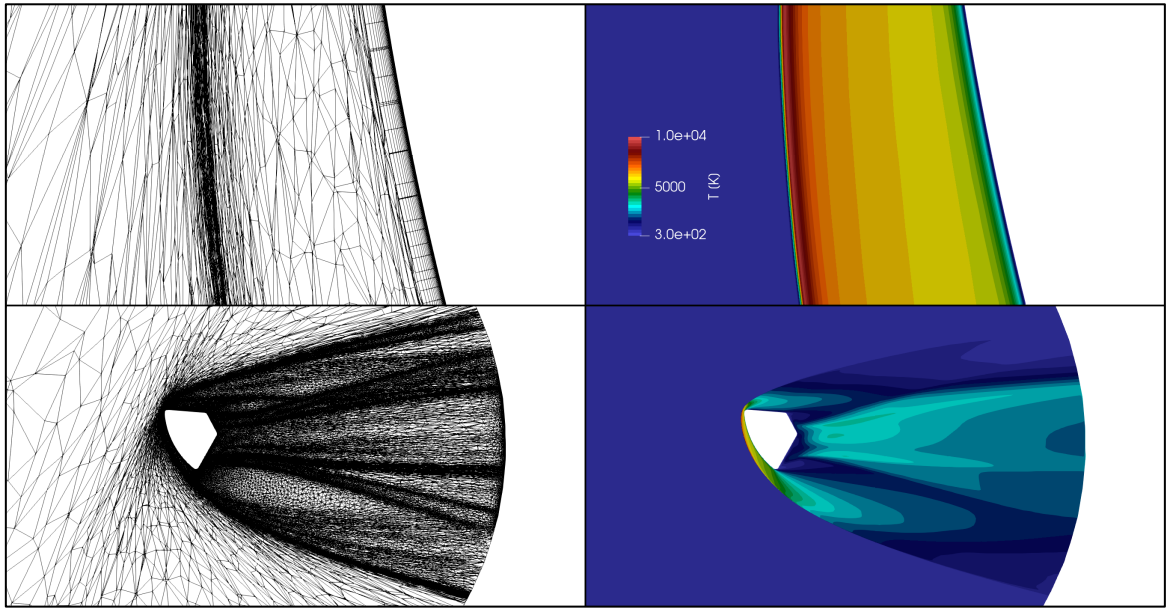
Crew Exploration Vehicle

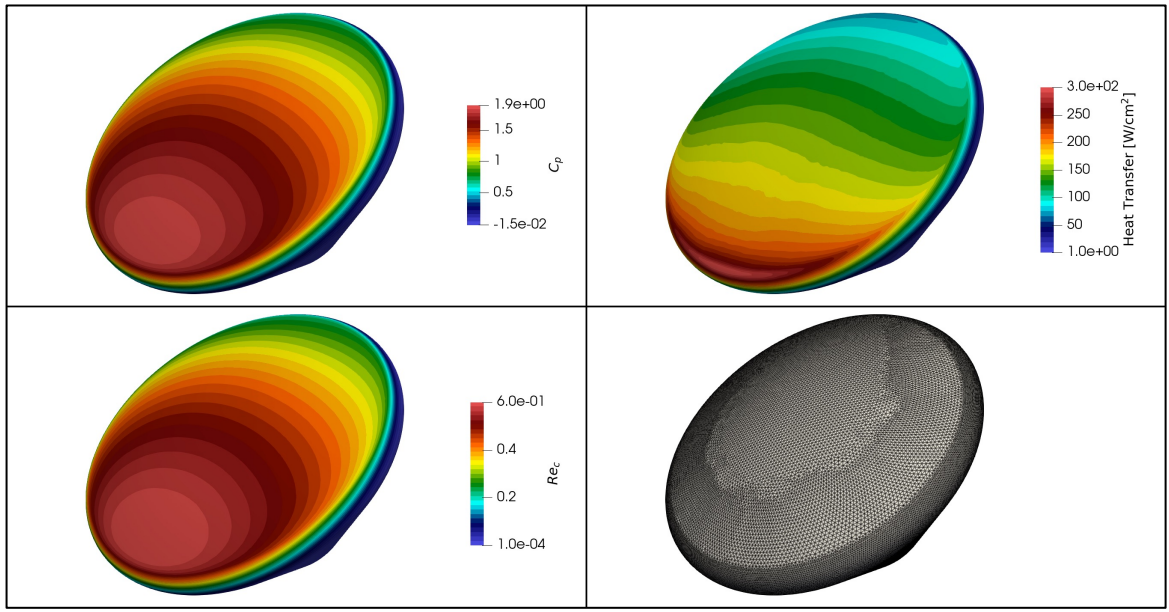
- Please see paper for details
- See past work for details on approach[1]
- Fixed thin prismatic BL and surface grid, with tetrahedral adaptation using NASA *refine* elsewhere
- $u_\infty = 4.6\frac{km}{s}, \alpha = 28^\circ$
- Laminar flow, 5-species air, two-temperature model
- Comparisons to DPLR and experiment available
- 3M point, 12M mixed elements for final grid
- 15 CFD and refinement cycles





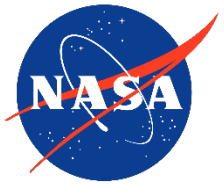






[1]Nastac, G., Tramel, R. W., & Nielsen, E. J. (2022). Improved Heat Transfer Prediction for High-Speed Flows over Blunt Bodies using Adaptive Mixed-Element Unstructured Grids. AIAA Paper 2022-0111.

Crew Exploration Vehicle



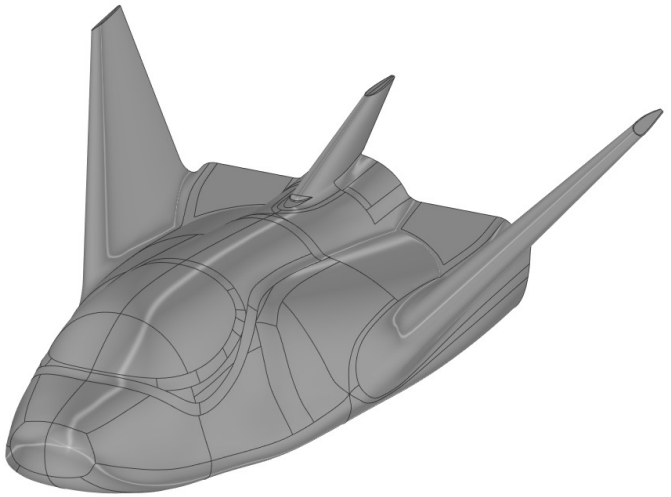
- NASA *refine* runs on CPUs and is used through files at this time
- I/O includes preprocessing and postprocessing
- GPU-enabled refinement will allow this approach to run more efficiently on GPU hardware
- A100 GPU results use 8 MPI ranks per GPU (generally 5-10% compute overhead while speeding up I/O)
- **This simulation is possible on a single node with one A100 GPU in under 6 hours**
  - Thermochemical nonequilibrium flow over a 3D capsule including wake and refinement with unstructured adapted grids

Architecture	Implementation	Total [min]	*refine* [min]	I/O [min]	CFD [min]	Speedup	Hardware MBW Ratio
15 AMD EPYC 7742 (1,920 cores)	Fortran	126	11	11	104	0.76	1.00
15 AMD EPYC 7742 (1,920 cores)	FLUDA	101	11	11	79	1.00	1.00
8 NVIDIA V100	FLUDA	162	82	21	59	2.51	2.20
4 NVIDIA 40 GB A100	FLUDA	150	57	14	79	3.75	3.80
1 NVIDIA 80 GB A100	FLUDA	337	57	18	262	4.52	4.72

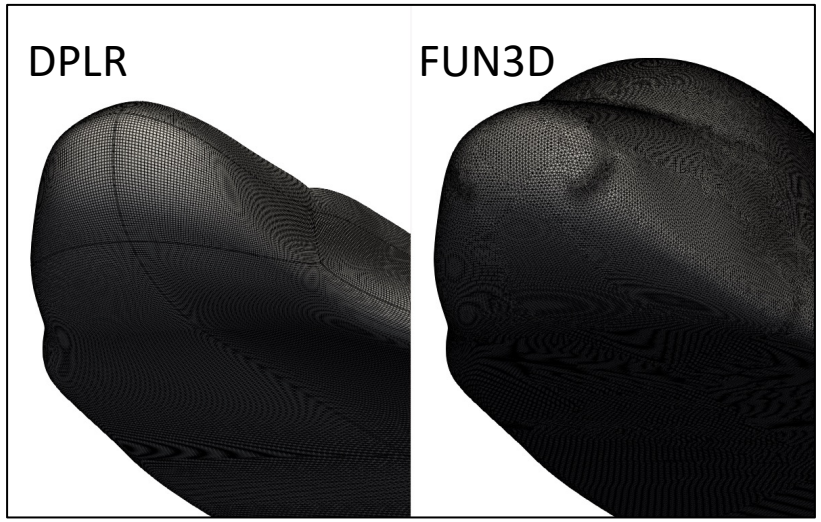
CFD performance scales with memory bandwidth ratio

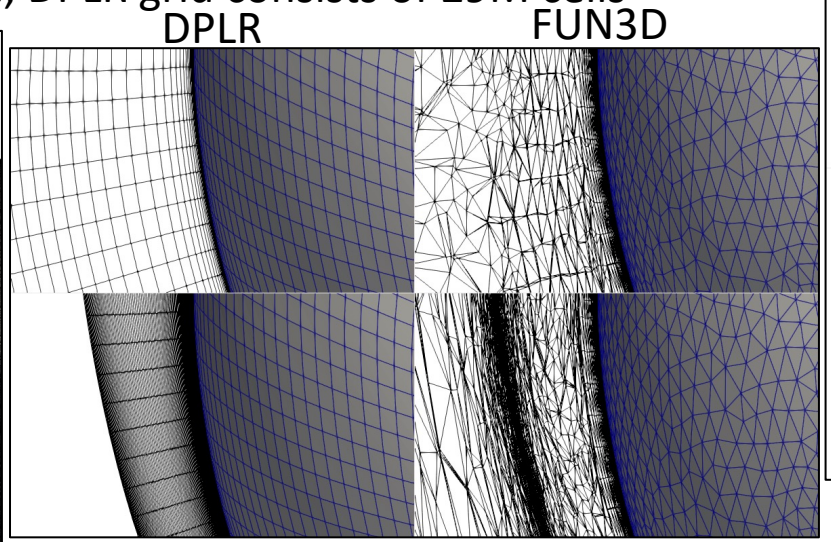
Speedup = Normalized hardware device-level performance
1 80 GB A100 ~ 4.52 EPYC nodes ~ 579 cores

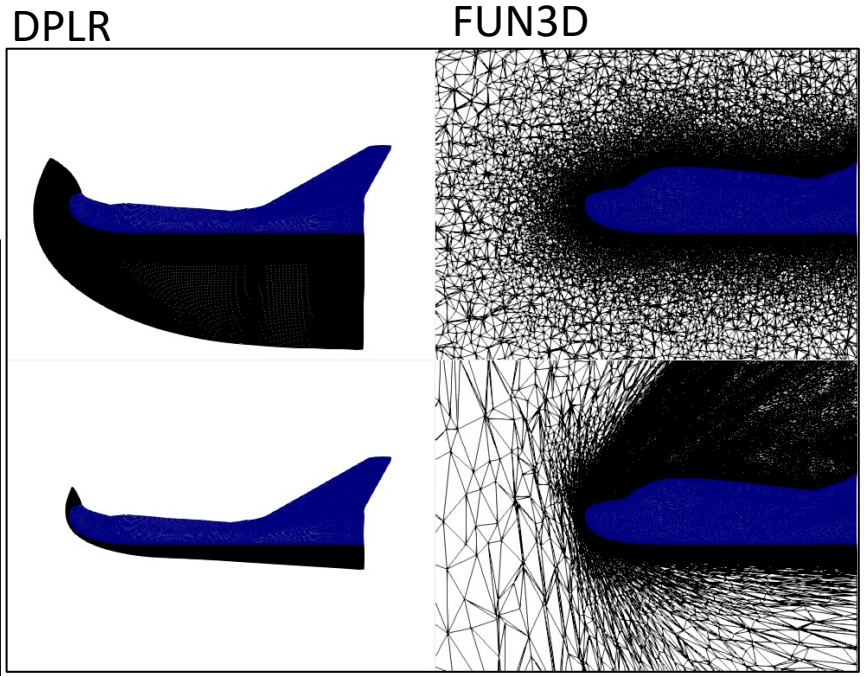
Sierra Space Dream Chaser©



- Please see paper for details
- Fixed thin prismatic BL and surface grid, with tetrahedral adaptation using NASA *refine* elsewhere
- $u_\infty = 5.0\frac{km}{s}, \alpha = 40^\circ$
- Wall is modeled as reaction cured glass (RCG) coating and assumed in radiative thermal equilibrium
- Laminar flow, 5-species air, one-temperature model
- Final unstructured grid contains 12M points, 50M mixed elements
- 15 CFD and refinement cycles
- Comparisons to NASA DPLR available, DPLR grid consists of 29M cells

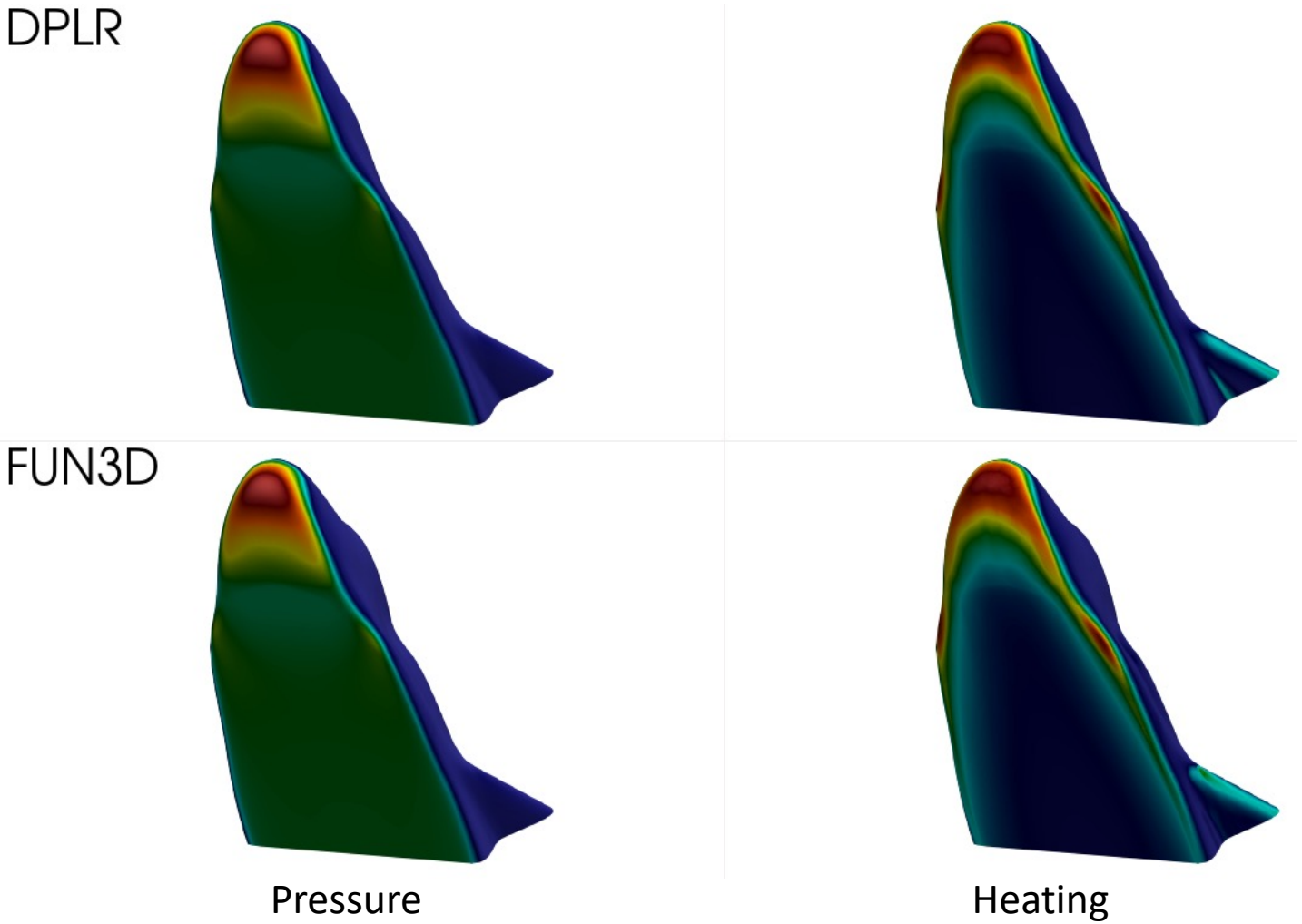


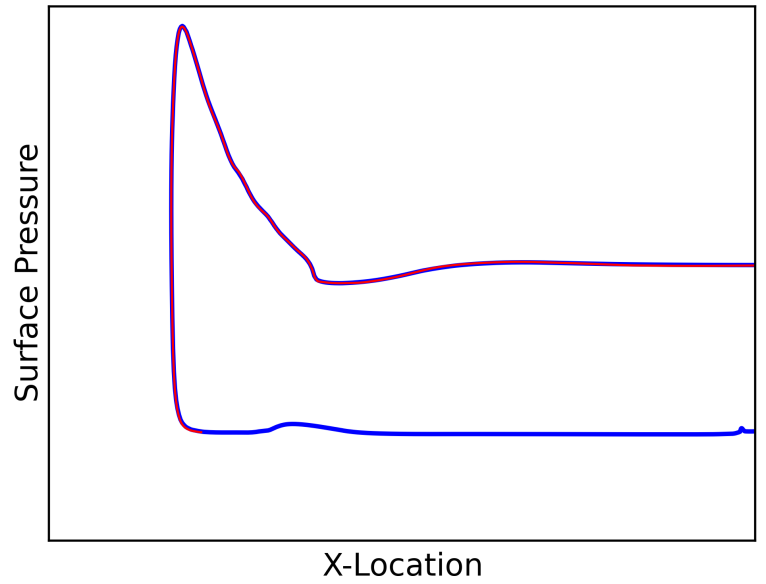


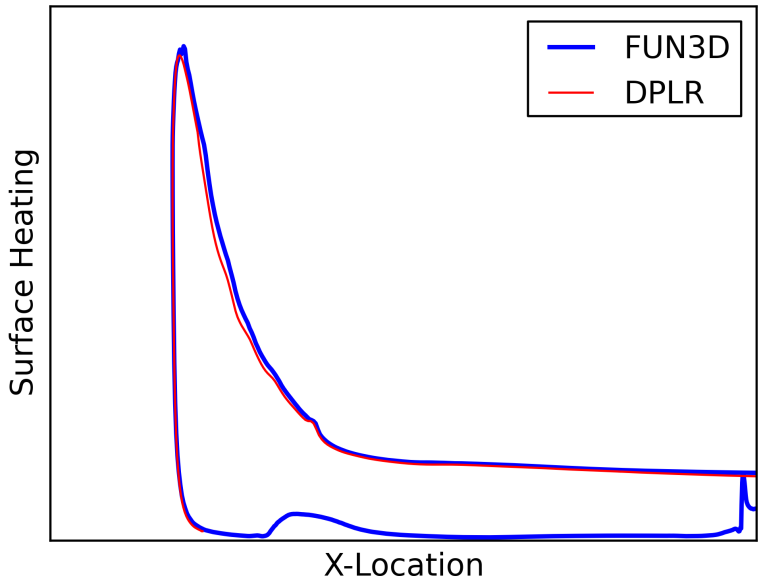


Credit: Sierra Space Corporation

Sierra Space Dream Chaser© (cont.)







At stagnation point:
Pressure within 0.2%
Heating within 2.0%

Credit: Sierra Space Corporation

Sierra Space Dream Chaser© (cont.)

- For GPU runs, grid refinement is a bottleneck due to running on CPU
- Current practice is to run refinement on separate CPU nodes, which complicates simulation process

Architecture	Implementation	Total [min]	*refine* [min]	I/O [min]	CFD [min]	Speedup	Hardware MBW Ratio
30 AMD EPYC 7742 (3,840 cores)	Fortran	132	24	7	101	0.83	1.00
30 AMD EPYC 7742 (3,840 cores)	FLUDA	115	24	7	84	1.00	1.00
16 NVIDIA V100	FLUDA	293	180	39	74	2.13	2.20

Speedup = Normalized hardware device-level performance
1 V100 ~ 2.13 EPYC nodes ~ 273 cores

CFD performance scales with memory bandwidth ratio

Code	DOFs	Iterations	refinement [min]	CFD [min]	Total [min]
DPLR	28.6M	7,000	6	52	58
FLUDA CPU	12.2M	21,000	24	91	115

FUN3D speed is comparable to DPLR for these simulations

Credit: Sierra Space Corporation

Summary

- Multi-architecture approach for implicit CFD on unstructured grids has been described and demonstrated for variety of problems ranging from low subsonic WMLES of aircraft to reentry vehicles
- Thin abstraction layer similar to CUDA C++ and HIP employed
  - ~500 lines of code of preprocessing macros
  - GPU-optimized code run on CPU with single thread
- Currently supports NVIDIA, AMD, Intel GPUs, and x86/ARM multicore CPUs
- Diverging code comprises less than 2% of total lines of code
- **CPUs better tolerate GPU-oriented code than the reverse for most of our application kernels**
- **Performance scales with memory bandwidth to first order as expected**
- **Performant GPU-enabled software enables design cycles and database generation to occur more quickly for less cost**
  - **Perfect gas RANS simulations on 4M point grids in minutes on 1 GPU**

Thanks to all of our collaborators and partners that helped make this happen!

This research was sponsored by the NASA Transformational Tools and Technologies (TTT) Project of the Transformative Aeronautics Concepts Program under the Aeronautics Research Mission Directorate