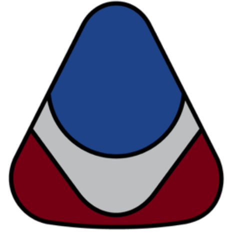


MIT–DOE/NNSA
Center for the Exascale Simulation of
Coupled High-Enthalpy
Fluid–Solid Interactions

Progress & Challenges of Multiphysics at Exascale: Thermo-Chemo-Mechanics Response of Hypersonics Thermal Protection Systems

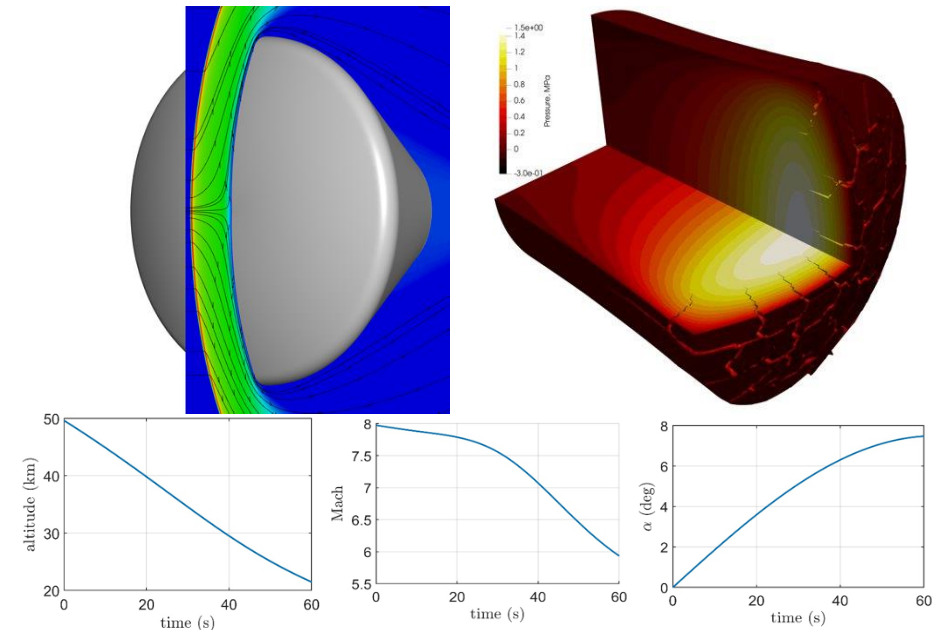
Raul Radovitzky
William Moses

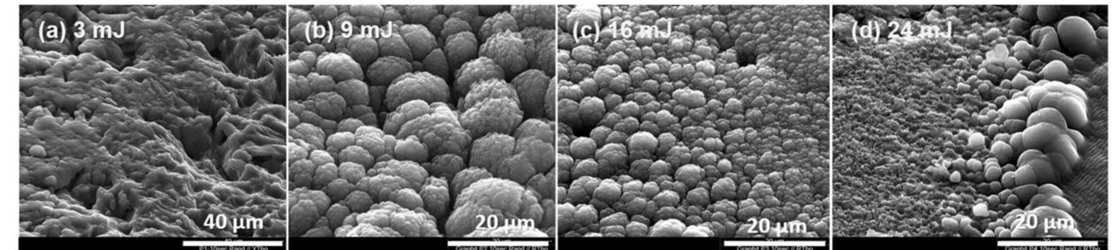
MICDE Predictive Science Symposium

April 15, 2026

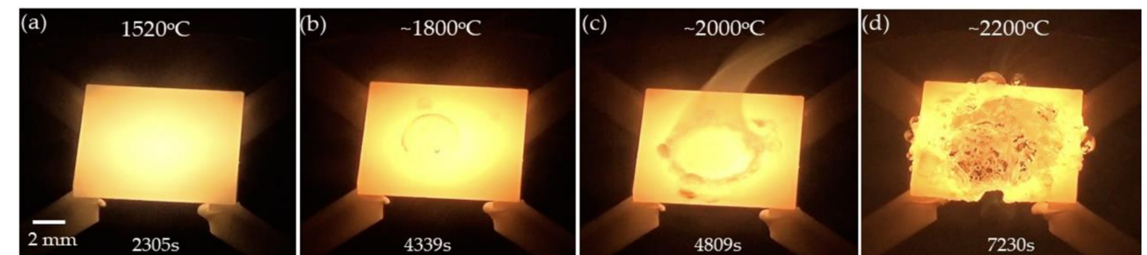
Our overarching application

- Predictive exascale simulation of a complete model heat shield system during atmospheric reentry along a prescribed trajectory
- Fully coupled end-to-end **aero-thermo-chemo-mechanics** response, with focus on material degradation and failure: C ablation, SiC oxidation/nitridation
- Novel AI/ML, UQ, and CS to enable full-physics exascale simulations





SEM images of graphite targets irradiated with four different pulse energies: (**a**) 3 mJ, (**b**) 9 mJ, (**c**) 16 mJ and (**d**) 24 mJ. Irradiation time: 10 s/150 pulses. Sierra-Trillo et al, 2022



Steam oxidation test for CVD-SiC/SiC composites using the laser heating facility (LAHF). Pham et al **2021**

Motivation for our overarching application

Unexpected failure modes in Orion Heat Shield TPS

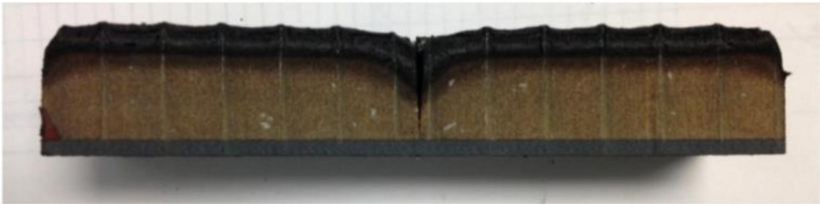
NASA Identifies Cause of Artemis I Orion Heat Shield Char Loss (12/05/2024)

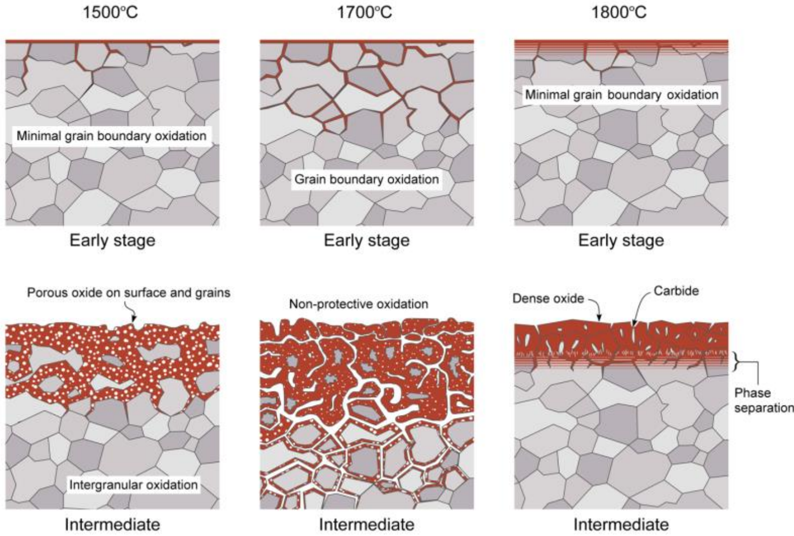
https://www.nasa.gov/missions/artemis/nasa-identifies-cause-of-artemis-i-orion-heat-shield-char-loss/

"… Using Avcoat material response data from Artemis I, the investigation team was able to replicate the Artemis I entry trajectory environment — a key part of understanding the cause of the issue — inside the arc jet facilities at NASA's Ames Research Center in California. They observed that during the period between dips into the atmosphere, heating rates decreased, and thermal energy accumulated inside the heat shield's Avcoat material. This led to the accumulation of gases that are part of the expected ablation process. Because the Avcoat did not have permeability, internal pressure built up, and led to cracking and uneven shedding of the outer layer…"

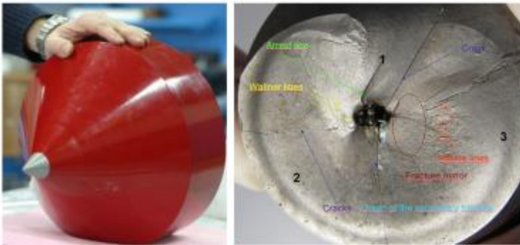
Wide range of material failure mechanisms caused by complex coupled thermo-chemo-mechanical processes

SHARP 2 cracking on leading edge strakes
(Sylvia Johnson, NASA Ames)

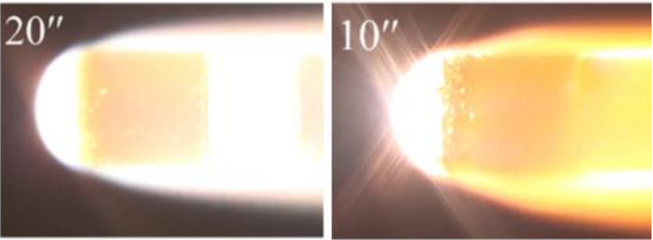

Deepened char and bond-coat heating around a fissure in AVCOAT
(Vander Kam & Gage 2021)



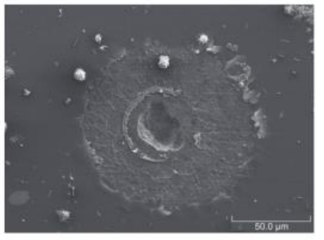
Backman et al. 2024

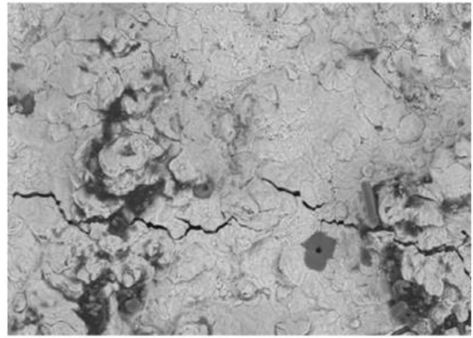

SHARK nose & post-flight crack surface (Saccone et al. 2011)

Pore pressure-induced crack in PR-CB
(Natali et al. 2016)



Passive & Active oxidation (Zhao et al. 2021)

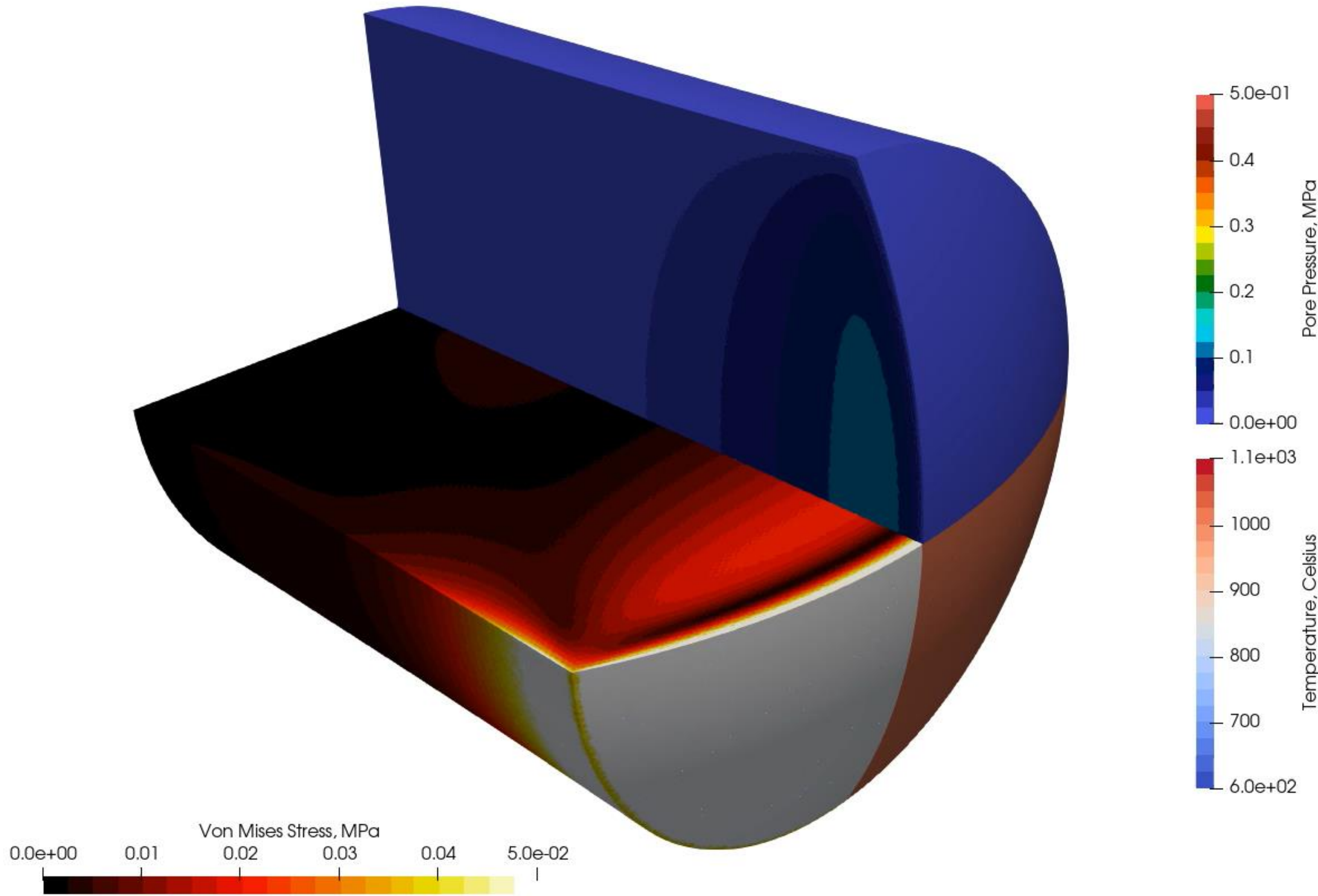

Oxide Delamination and Pitting
(Harder et al. 2013)


Plasma-induced oxidation and microcracking
(Cecere et al. 2015)

Pyrolysis of P50 Cork. Courtesy of Prof. Koo and his TPS research group at UT Austin.

CHEFSI

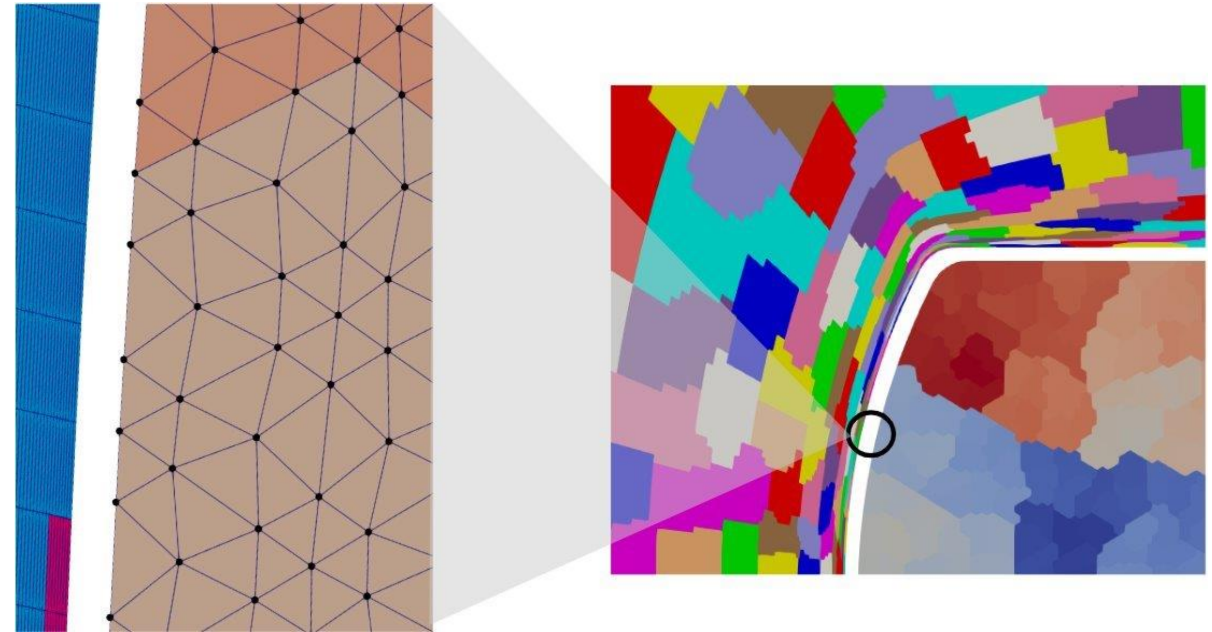
Large-Scale Simulations: 370 Million Degrees of Freedom on Dane



- Monolithic thermochemistry
  - Two component decomposition (Goldstein, 1965)
  - Newton's method with PETSc
  - Additive Schwartz & GAMG on 1120 subdomains
- Solid mechanics
  - Linear poroelasticity (Terzaghi, 1925)
  - Fracture (Pickard, 2026)
  - Dynamic relaxation solver (Pickard et al. 2023)
- 7.4 Million 10-node tetrahedral elements (5 DOF/node)

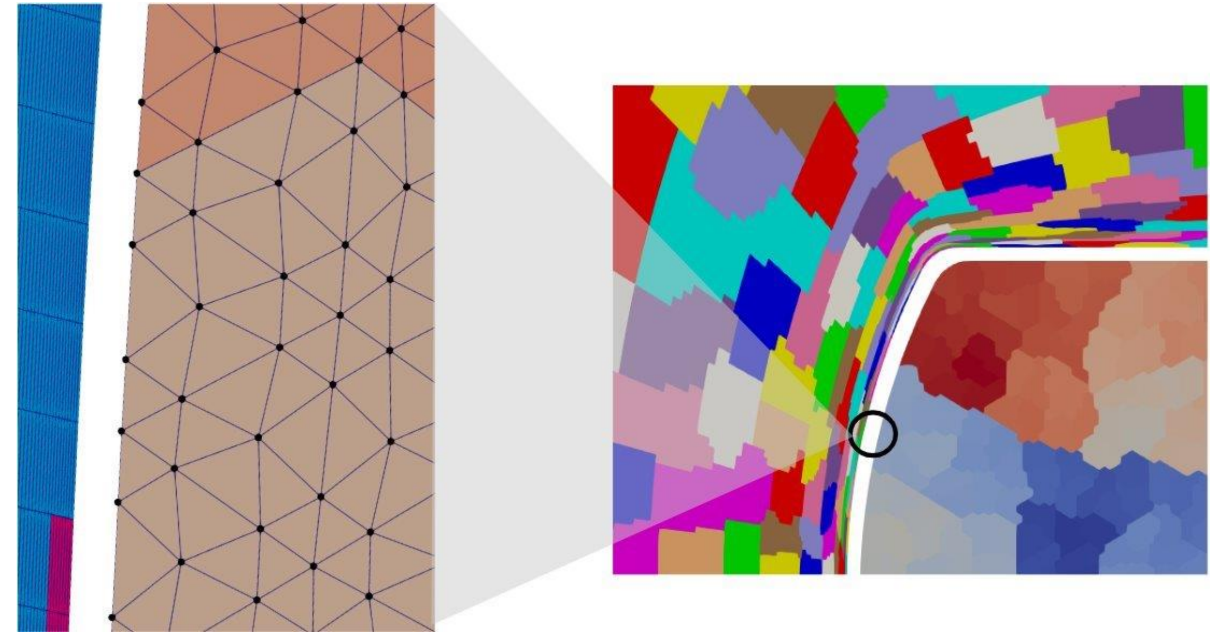
Scaling Challenges in Different Axes

- Resolution
  - Make the simulation time and memory efficient
  - Leverage clusters of accelerators
- Complexity
  - Multiphysics coupling, scientific equations
  - Non-matching meshes and partitions
  - Multiscale material response: AI/ML (surrogates)
- Multiple Simulations
  - Predictive algorithms (like UQ) require many runs
  - Gradients are required!



Scaling Challenges in Different Axes

- Resolution
  - Make the simulation time and memory efficient
  - Leverage clusters of accelerators
- Complexity
  - Multiphysics coupling, scientific equations
  - Non-matching meshes and partitions
  - Multiscale material response: AI/ML (surrogates)
- Multiple Simulations
  - Predictive algorithms (like UQ) require many runs
  - Gradients are required!



CHEFSI Simulation Stack

CHEFSI

e3nn

ML interatomic potentials

classical MD (atomistic scale)

LAMMPS

microstructure simulations (mesoscale)

precomputed vehicle trajectory

& SPARTA

aero-thermo-chemistry

MIT / CHEFSI KITESurF

gas-material interaction model

e3nn upscaling

e3nn, MDR upscaling

thermo-chemo-mechanics

Pervasive VVUQ

aero-thermo-chemo-mechanics

Languages

julia

SmartSolve

DAGGER Solvers

OpenCilk

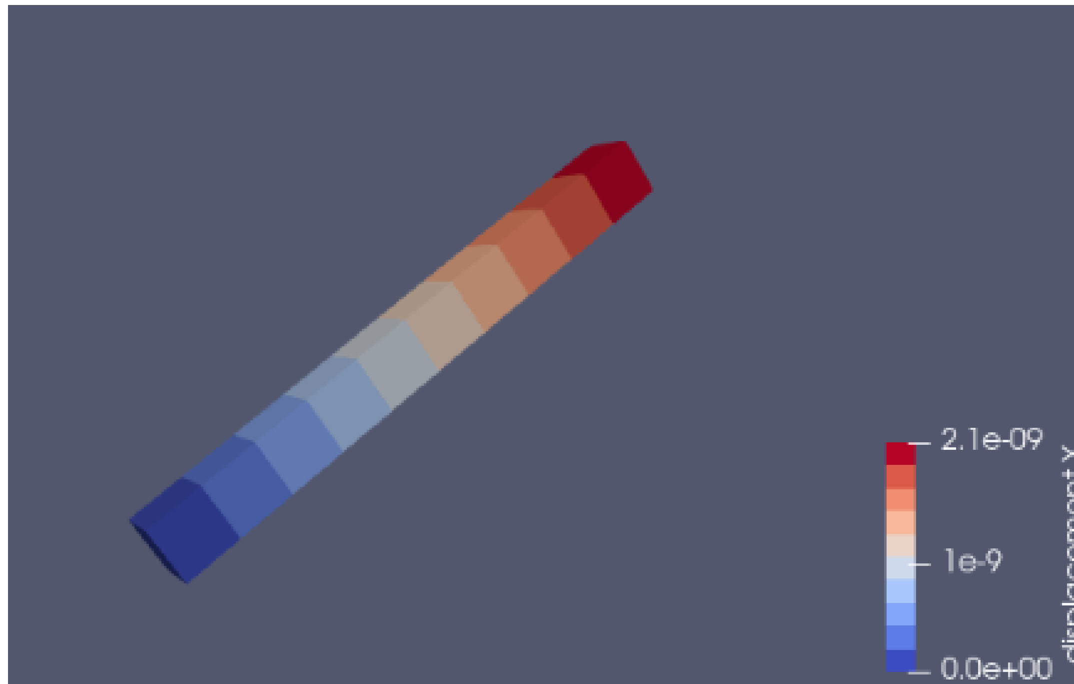
Compilers and runtimes

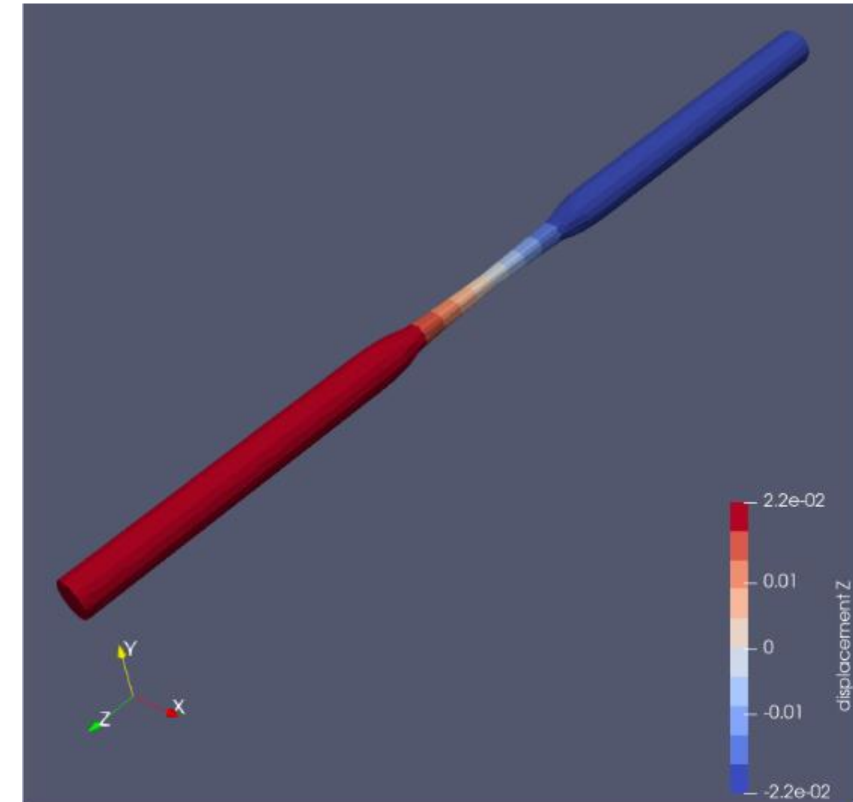
Exascale hardware

TUOLUMNE

ΣMIT on GPU using Kokkos

- Ported Explicit Dynamics and Dynamic Relaxation solvers.
- Speedup of 30X for Linear Elastic.
- Less gain in plasticity code (13X for J2Linear) due to warp divergence. Not sure how to port log and exp of matrices for updates in large deformation plasticity.
- Got ΣMIT running on Dane.





ΣMIT on GPU using Kokkos

Problem: Porting just this part of ΣMIT's codebase required almost fully rewriting it.

- Approximately 7500 lines added or modified in the main source files alone.
- Porting more of ΣMIT's solvers, systems, materials, etc. will require more rewriting.
- Even with AI coding agents, such extensive rewriting is risky.

Can we avoid such extensive rewriting?

Git statistics for ΣMIT Kokkos branch

```
src/elements/element_set_body.cc                             |  231 ------------
src/elements/element_set_body.h                              |   19 +-
src/elements/element_set_device.cc                           |  158 +++++++++
src/elements/element_set_device.h                            |  137 ++++++++
src/elements/element_set_interface_lubrication.cc            |    6 +-
src/elements/element_set_interface_tetP.cc                   |    2 +-
src/elements/element_set_interface_triP.cc                   |    2 +-
src/elements/element_set_lubrication.cc                      |    2 +-
src/elements/element_set_manifold_TriP.cc                    |    2 +-
src/elements/element_set_monolithic_interface_tetP.h         |    2 +-
src/elements/element_set_monolithic_tetP_tetP.h              |    2 +-
src/elements/element_set_one_sided_lubrication.cc            |    4 +-
src/elements/element_set_shellP_shellP.cc                    |    4 +-
src/elements/element_set_shellP_tetQ.cc                      |    2 +-
src/elements/element_set_tetP.cc                             |  154 +++++---
src/elements/element_set_tetP.h                              |   26 +-
src/elements/element_set_tetP_shellQ.cc                      |    2 +-
src/elements/element_set_tetP_tetP.cc                        |    8 +-
src/elements/element_set_tri1_tri1.cc                        |    8 +-
src/elements/element_set_triP.cc                             |  150 +++++---
src/elements/element_set_triP.h                              |   22 ++
src/elements/element_set_triP_triP.cc                        |    8 +-
src/elements/element_set_util.h                              |   51 +++
src/elements/element_set_util.icc                            |  246 +++++++++++++
src/elements/element_set_weak_boundary_shell.cc              |    2 +-
src/fem/nodal_field.h                                        |   76 +++-
src/fem/nodal_field.icc                                      |  142 ++++++--
src/io/vtk/mesh_writer_vtk.cc                                |    3 -
src/kokkos/elements/CMakeLists.txt                           |   44 +++
src/kokkos/elements/kokkos_element_set_triP.cc               |  574 ++++++++++++++++++++++++++++++
src/kokkos/elements/kokkos_element_set_triP.h                |  279 +++++++++++++++
src/kokkos/elements/test_element_set_triP_kokkos.cc          |  199 +++++++++++
src/kokkos/materials/my_kokkos_elastic.h                     |  148 ++++++++
src/kokkos/materials/my_kokkos_elastic.icc                   |  342 ++++++++++++++++++
src/kokkos/weakforms/CMakeLists.txt                          |   47 +++
src/kokkos/weakforms/my_kokkos_region.cc                     |  668 +++++++++++++++++++++++++++++++++++
src/kokkos/weakforms/my_kokkos_region.h                      |  598 +++++++++++++++++++++++++++++++
src/kokkos/weakforms/test_kokkos_region.cc                   |  255 ++++++++++++++
src/materials/J2_linear/J2_linear.h                          |   11 +
src/materials/J2_linear/J2_linear_device.h                   |  671 +++++++++++++++++++++++++++++++++++
src/materials/elastic/elastic.cc                             |  261 +++++++++++---
src/materials/elastic/elastic.h                              |   57 ++-
src/materials/elastic/elastic_device.h                       |  261 ++++++++++++++
src/materials/material_consistency_test.cc                   |   34 +-
src/materials/mechanical_material.cc                         |   20 ++
src/materials/mechanical_material.h                          |   23 ++
src/materials/reaction_diffusion/lin_conv_diff.cc            |   22 +-
src/materials/reaction_diffusion/lin_conv_diff.h             |   14 +-
src/materials/reaction_diffusion/lin_conv_diff_device.h      |   83 +++++
src/materials/reaction_diffusion/rxndiff_consistency_test.cc |   48 +--
src/mathlib/{mat3.cc => mat3.icc}                            |   22 +-
src/mathlib/mathlib.h                                        |   63 ++--
src/mathlib/{mathlib.cc => mathlib.icc}                      |   17 +-
src/mathlib/mathmat.h                                        |    8 +
src/mathlib/mathmat.icc                                      |    8 +
src/mathlib/sparse/sparse_crs.cc                             |    2 +-
src/mathlib/sparse/sparse_crs.h                              |   19 +-
src/mathlib/sparse/sparse_matrix.h                           |    2 +-
src/mathlib/sparse/sparse_skyline.cc                         |    2 +-
src/mathlib/sparse/sparse_skyline.h                          |   14 +-
src/mathlib/sparse/sparse_solver.cc                          |   13 +-
src/mathlib/sparse/sparse_solver_cudss.cu                    |  168 +++++++++
src/mathlib/sparse/sparse_solver_cudss.h                     |   56 +++
src/mesh/mesh.cc                                             |   11 +-
src/solvers/dynamic_relaxation.cc                            |  350 +++++++++++++++---
src/solvers/dynamic_relaxation.h                             |   15 +-
src/solvers/explicit_newmark_integrator.cc                   |  296 +++++++++++++---
src/solvers/explicit_newmark_integrator.h                    |   14 +-
src/solvers/solver.cc                                        |    6 +-
src/summit_base.h                                            |    1 +
src/summit_enum.h                                            |    1 +
src/systems/dynamics_system.cc                               |   12 +-
src/systems/matrix_free_system.cc                            |    4 +-
src/utils/device_macros.h                                    |   36 ++
src/utils/device_memory.h                                    |  183 ++++++++++
src/utils/math-util.h                                        |    3 +
src/utils/math-util.icc                                      |    1 +
src/utils/test-utils.h                                       |   39 --
src/weakforms/mechanics_region.cc                            | 1191 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++-
src/weakforms/mechanics_region.h                             |   56 +++
src/weakforms/reaction_diffusion_region.cc                   |  254 ++++++++++++++
src/weakforms/reaction_diffusion_region.h                    |   30 ++
```

ΣMIT on GPU using Kitsune

Kitsune NVIDIA & AMD GPU support

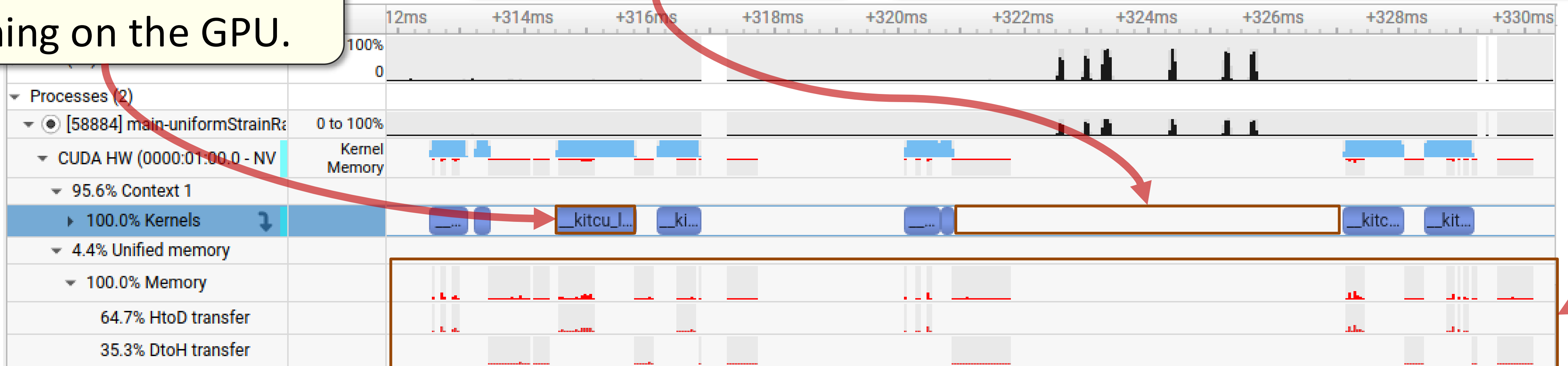
We used Kitsune to make part of this Cilk parallelization run on a GPU just by adding some compiler flags.

Some early success, but performance needs work.

Some parallel loops are running on the GPU.

Some computation is still on CPU.

Moving data between CPU and GPU hurts the performance of both.



Nsight profile on local test machine.

Current GPU-ΣMIT blockers

```
void summit::commonInterfaceMechanicsRegion::Residual(
    NodalField<real> const& u, NodalField<real> const& u0,
    real dt, bool update, CommunicationManager const& commManager,
    NodalField<real>& residual) {
  // ...
  // allocate memory for the local displacement (previous and current)
  static std::vector<real> _ul;
  _ul.resize(residual_dim);
  cilk::span_holder<real> ul(_ul);
  static std::vector<real> _u0l;
  _u0l.resize(residual_dim);
  cilk::span_holder<real> u0l(_u0l);
  static std::vector<real> _rl;
  _rl.resize(residual_dim);
  cilk::span_holder<real> rl(_rl);
  // ****************** PART I: Prepare residua
  // loop over the elements of the element set
  #pragma cilk grainsize 1
  cilk_for (size_t ei(0); ei != _element_set->el
    elem_t e(ei);
    // extract local part of unknown
    _element_set->Localize(u, e, ul);
    _element_set->Localize(u0, e, u0l);
    // compute elementary residual
    ElementaryConstitutive(e, dt, update, ul, u0l,
                           Pavg_JumpU.local(e, quad_t(0)));
  }
  // ...
}
```

```
void summit::InterfaceMechanicsRegion::ElementaryConstitutive(
    elem_t e, real dt, bool update, std::span<real> const ul,
    std::span<real> const u0l, real* TJ) {
  // ...
  // integrate over quadrature points
  for (quad_t q(0); q < _element_set->nquad(); ++q) {
    // constitutive update of the left element
    LeftConstitutiveUpdate(e, q, ul, dt, P_L_new_span, F_L_new_span,
                           internal_L_new_span, _C_L_red.data());
    // constitutive update of the right element
    RightConstitutiveUpdate(e, q, ul, dt, P_R_new_span, F_R_new_span,
                            internal_R_new_span, _C_R_red.data());
    // ...
  }
}
```

What's needed to make this parallelization work on GPUs?

Support parallel reductions

Deal with virtual functions

Future problem: Memory footprint for larger simulations

Computing Hardware is No Longer For Everybody

Computing Hardware is No Longer For Everybody

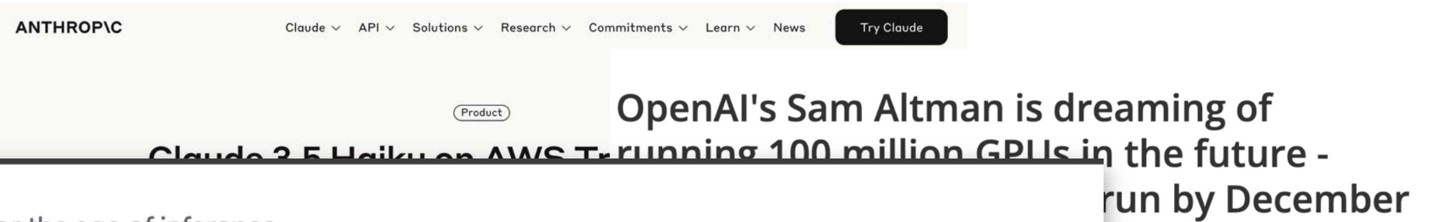
CHEFSI

Exclusive: Meta begins testing its first in-house AI training chip

By Katie Paul and Krystal Hu

March 11, 2025 2:37 PM GMT+1 · Updated March 11, 2025

[1/2] The logo of Meta Platforms' business group is seen in Brussels, Belgi... Herman/File Photo Purchase Licensing Rights



OpenAI's Sam Altman is dreaming of running 100 million GPUs in the future - ...run by December

...mething to think about

Ironwood: The first Google TPU for the age of inference

- When scaled to 9,216 chips per pod for a total of 42.5 Exaflops, Ironwood supports more than 24x the compute power of the world's largest supercomputer – El Capitan – which offers just 1.7 Exaflops per pod. Ironwood delivers the massive parallel processing power necessary for the most demanding AI workloads, such as super large size dense LLM or MoE models with thinking capabilities for training and inference. Each individual chip boasts peak compute of 4,614 TFLOPs. This represents a monumental leap in AI capability. Ironwood's memory and network architecture ensures that the right data is always available to support peak performance at this massive scale.

NVIDIA Puts Grace Blackwell on Every Desk and at Every AI Developer's Fingertips

NVIDIA Project DIGITS With New GB10 Superchip Debuts as World's Smallest AI Supercomputer Capable of Running 200B-Parameter Models

more Nvidia GPUs, an impulse all PC gamers can truly understand

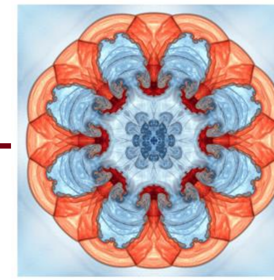
News By Andy Edser published 23 July 2025

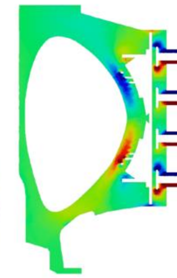
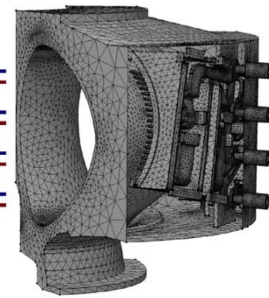
I've checked down the back of the sofa, and I'm not sure I can cover it.

Comments (2)

CHEFSI

Scientific Computing Today



- Scientists do not write TPU* code

```
__global__
void AddNodeForcesFromElems_kernel( Index_t numNode,
                                    Index_t padded_numNode,
                                    const Int_t* nodeElemCount,
                                    const Int_t* nodeElemStart,
                                    const Index_t* nodeElemCornerList,
                                    const Real_t* fx_elem,
                                    const Real_t* fy_elem,
                                    const Real_t* fz_elem,
                                    Real_t* fx_node,
                                    Real_t* fy_node,
                                    Real_t* fz_node,
                                    const Int_t num_threads)
{
    int tid=blockDim.x*blockIdx.x+threadIdx.x;
    if (tid < num_threads)
    {
      Index_t g_i = tid;
      Int_t count=nodeElemCount[g_i];
      Int_t start=nodeElemStart[g_i];
      Real_t fx,fy,fz;
      fx=fy=fz=Real_t(0.0);

      for (int j=0;j<count;j++)
      {
          Index_t pos=nodeElemCornerList[start+j]; // Uncoalesced access here
          fx += fx_elem[pos];
          fy += fy_elem[pos];
          fz += fz_elem[pos];
      }


      fx_node[g_i]=fx;
      fy_node[g_i]=fy;
      fz_node[g_i]=fz;
    }
}
```

CHEFSI

Scientific Computing Today

- Scientists do not write TPU* code
  - BIG (MFEM library alone is 737K LOC)

```
__global__
void AddNodeForcesFromElems_kernel( Index_t numNode,
                                    Index_t padded_numNode,
                                    const Int_t* nodeElemCount,
                                    const Int_t* nodeElemStart,
                                    const Index_t* nodeElemCornerList,
                                    const Real_t* fx_elem,
                                    const Real_t* fy_elem,
                                    const Real_t* fz_elem,
                                    Real_t* fx_node,
                                    Real_t* fy_node,
                                    Real_t* fz_node,
                                    const Int_t num_threads)
{
    int tid=blockDim.x*blockIdx.x+threadIdx.x;
    if (tid < num_threads)
    {
      Index_t g_i = tid;
      Int_t count=nodeElemCount[g_i];
      Int_t start=nodeElemStart[g_i];
      Real_t fx,fy,fz;
      fx=fy=fz=Real_t(0.0);

      for (int j=0;j<count;j++)
      {
          Index_t pos=nodeElemCornerList[start+j]; // Uncoalesced access here
          fx += fx_elem[pos];
          fy += fy_elem[pos];
          fz += fz_elem[pos];
      }


      fx_node[g_i]=fx;
      fy_node[g_i]=fy;
      fz_node[g_i]=fz;
    }
}
```

Scientific Computing Today

- Scientists do not write TPU* code
  - BIG (MFEM library alone is 737K LOC)
  - Templated

```
__global__
void AddNodeForcesFromElems_kernel( Index_t numNode,
                                    Index_t padded_numNode,
                                    const Int_t* nodeElemCount,
                                    const Int_t* nodeElemStart,
                                    const Index_t* nodeElemCornerList,
                                    const Real_t* fx_elem,
                                    const Real_t* fy_elem,
                                    const Real_t* fz_elem,
                                    Real_t* fx_node,
                                    Real_t* fy_node,
                                    Real_t* fz_node,
                                    const Int_t num_threads)
{
    int tid=blockDim.x*blockIdx.x+threadIdx.x;
    if (tid < num_threads)
    {
      Index_t g_i = tid;
      Int_t count=nodeElemCount[g_i];
      Int_t start=nodeElemStart[g_i];
      Real_t fx,fy,fz;
      fx=fy=fz=Real_t(0.0);

      for (int j=0;j<count;j++)
      {
          Index_t pos=nodeElemCornerList[start+j]; // Uncoalesced access here
          fx += fx_elem[pos];
          fy += fy_elem[pos];
          fz += fz_elem[pos];
      }


      fx_node[g_i]=fx;
      fy_node[g_i]=fy;
      fz_node[g_i]=fz;
    }
}
```

CHEFSI

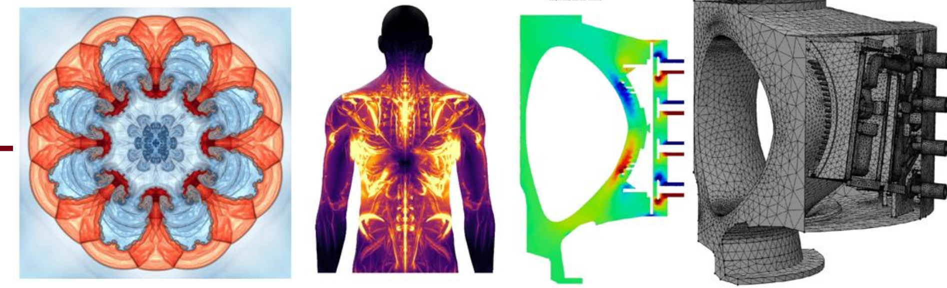
Scientific Computing Today

- Scientists do not write TPU* code
  - BIG (MFEM library alone is 737K LOC)
  - Templated
  - Not in Python

```
__global__
void AddNodeForcesFromElems_kernel( Index_t numNode,
                                    Index_t padded_numNode,
                                    const Int_t* nodeElemCount,
                                    const Int_t* nodeElemStart,
                                    const Index_t* nodeElemCornerList,
                                    const Real_t* fx_elem,
                                    const Real_t* fy_elem,
                                    const Real_t* fz_elem,
                                    Real_t* fx_node,
                                    Real_t* fy_node,
                                    Real_t* fz_node,
                                    const Int_t num_threads)
{
    int tid=blockDim.x*blockIdx.x+threadIdx.x;
    if (tid < num_threads)
    {
      Index_t g_i = tid;
      Int_t count=nodeElemCount[g_i];
      Int_t start=nodeElemStart[g_i];
      Real_t fx,fy,fz;
      fx=fy=fz=Real_t(0.0);

      for (int j=0;j<count;j++)
      {
          Index_t pos=nodeElemCornerList[start+j]; // Uncoalesced access here
          fx += fx_elem[pos];
          fy += fy_elem[pos];
          fz += fz_elem[pos];
      }


      fx_node[g_i]=fx;
      fy_node[g_i]=fy;
      fz_node[g_i]=fz;
    }
}
```

CHEFSI

Scientific Computing Today



- Scientists do not write TPU* code
  - BIG (MFEM library alone is 737K LOC)
  - Templated
  - Not in Python
  - Sometimes* in CUDA

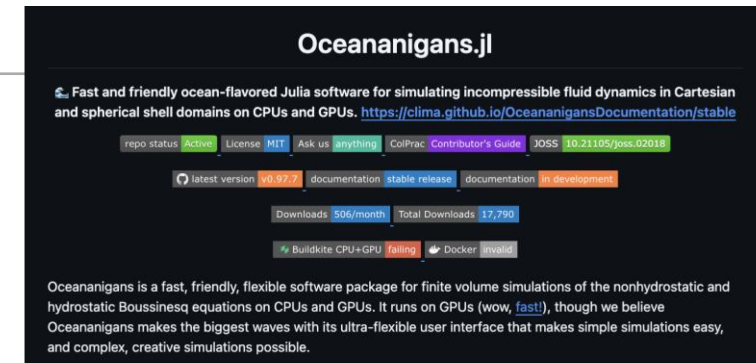
```
template <>
struct RajaCuWrap<3>
{
   template <const int BLCK = MFEM_CUDA_BLOCKS, typename DBODY>
   static void run(const int N, DBODY &&d_body,
                   const int X, const int Y, const int Z, const int G)
   {
      RajaCuWrap3D(N, d_body, X, Y, Z, G);
   }
};
```

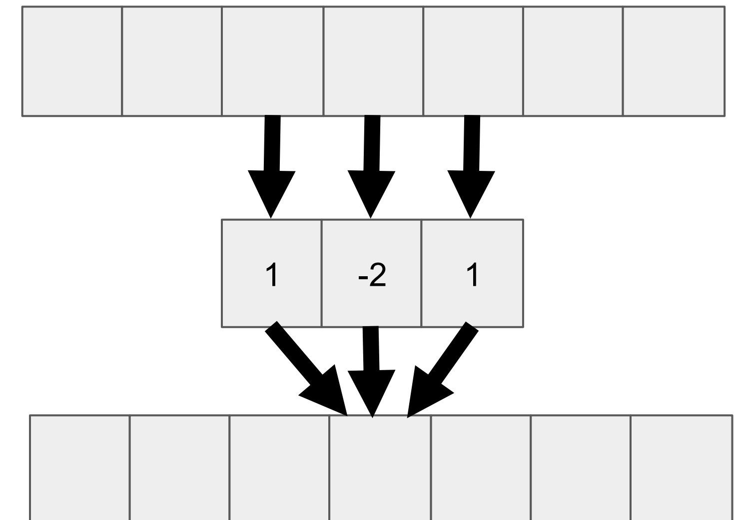
```
__global__
void AddNodeForcesFromElems_kernel( Index_t numNode,
                                    Index_t padded_numNode,
                                    const Int_t* nodeElemCount,
                                    const Int_t* nodeElemStart,
                                    const Index_t* nodeElemCornerList,
                                    const Real_t* fx_elem,
                                    const Real_t* fy_elem,
                                    const Real_t* fz_elem,
                                    Real_t* fx_node,
                                    Real_t* fy_node,
                                    Real_t* fz_node,
                                    const Int_t num_threads)
{
    int tid=blockDim.x*blockIdx.x+threadIdx.x;
    if (tid < num_threads)
    {
      Index_t g_i = tid;
      Int_t count=nodeElemCount[g_i];
      Int_t start=nodeElemStart[g_i];
      Real_t fx,fy,fz;
      fx=fy=fz=Real_t(0.0);

      for (int j=0;j<count;j++)
      {
          Index_t pos=nodeElemCornerList[start+j]; // Uncoalesced access here
          fx += fx_elem[pos];
          fy += fy_elem[pos];
          fz += fz_elem[pos];
      }


      fx_node[g_i]=fx;
      fy_node[g_i]=fy;
      fz_node[g_i]=fz;
    }
}
```

Looking More Deeply at Scientific Code

CHEFSI

```
function stencil_kernel(y, x)
  i = threadIdx().x + (blockIdx().x - 1) * blockDim().x
  if i <= length(x) - 2
    y[i] = x[i] - 2 * x[i + 1] + x[i + 2]
  end
end

function model(...)
  @cuda threads=... blocks=... stencil_kernel(y, x)
  @cuda threads=... blocks=... stencil_kernel(x, y)
end
```



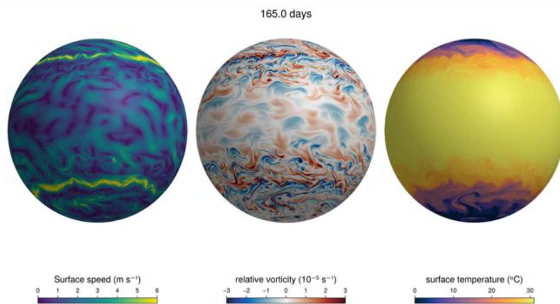


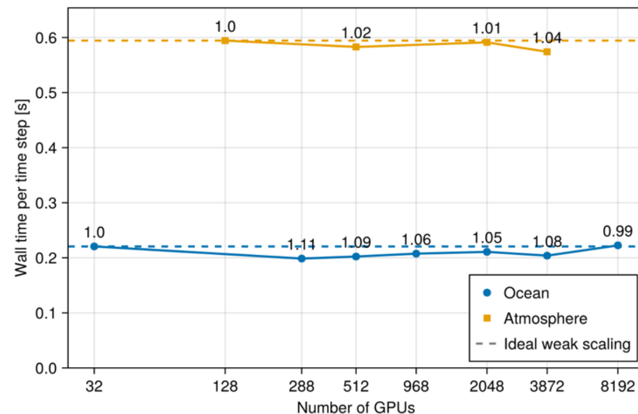
> 277 such kernels

CUDA To StableHLO

CHEFSI

- Automatically ported state-of-the-art single-device version of climate code
  - 8,192 NVIDIA GPUs (DOE Perlmutter, Alps)
  - 1,679 Google TPUs v6e (918 TFLOPS each)
- Communication optimizations are key!
- Good Single-Node Perf
  - Vanilla Model: 272.0seconds
  - Tensor Optims: 11.5seconds






```
function stencil_kernel(y, x)
  i = threadIdx().x + (blockIdx().x - 1) * blockDim().x
  if i <= length(x) - 2
    y[i] = x[i] - 2 * x[i+1] + x[i+2]
  end
end

function model(...)
  @cuda threads=... blocks=... stencil_kernel(y, x)
  @cuda threads=... blocks=... stencil_kernel(x, y)
end
```

Compilation

```
define void @julia_difference_kernel_890({}* %y, {}* %x) {
top:
  %3 = call i32 @llvm.nvvm.read.ptx.sreg.tid.x()
  %4 = add nuw nsw i32 %3, 1
  ...
  br i1 %.not, label %common.ret, label %L31
}
```

Raising

```
func.func @kernel(%y : memref<100xf64>, %x : memref<100xf64>) {
  affine.parallel %arg1 = 0 to 100 {
    %x1 = affine.load %x[%arg1]
    %x2 = affine.load %x[%arg1 + 1]
    ...
    affine.store %sum, %y[%arg1]
  }
}
```

Multi-Dimensionalization

```
%x1 = stablehlo.slice %x[1:98]
%x2 = stablehlo.slice %x[2:99]
%mul = stablehlo.multiply %x2, tensor<2.0>
%add = stablehlo.add %x1, %mu
```

Optimization

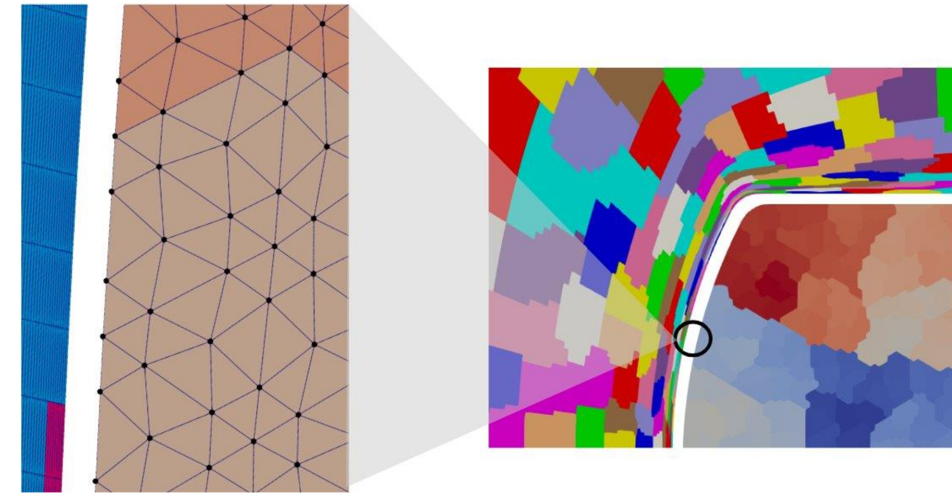
```
res = stablehlo.convolve %x, tensor<[1.0, -4.0, 6.0, -4.0, 1.0]>
```

Challenges in MultiPhysics Codes

CHEFSI

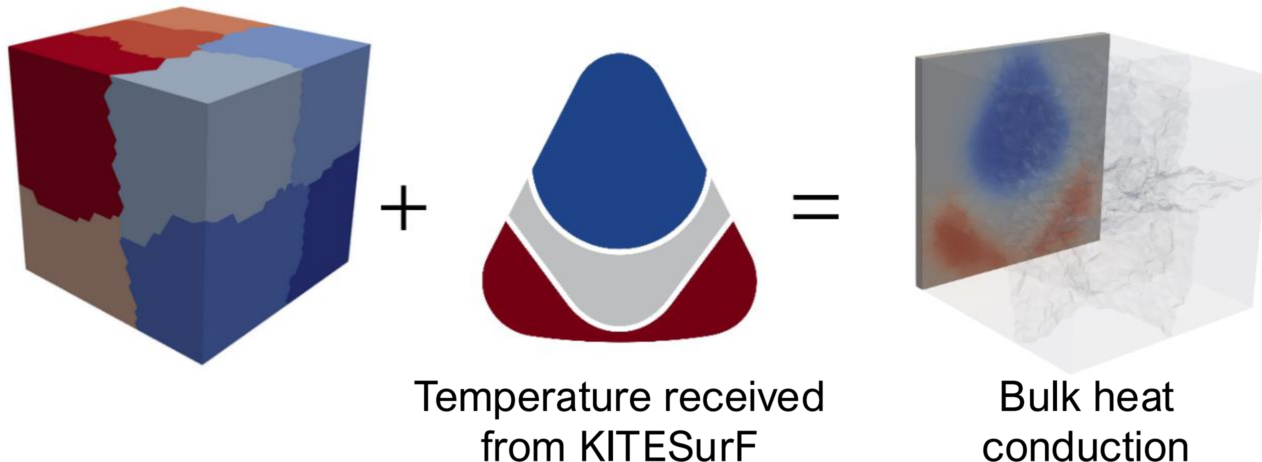
MIT / CHEFSI KITESurF

- More complex Multi-Physics doesn't match modern ML workflows
- ML frameworks cannot do irregular ghost region without padding
- Kinetic Interface Transport and Exchange on Surfaces (KITESURF)



Parallel partitioning with non-matching meshes

Demonstration of preliminary coupling of KITESurf with ΣMIT



Temperature received from KITESurF

Bulk heat conduction

Nonequilibrium gas surface interaction formalism (Marschall & MacLean)

Leverage ML Kernel DSLs without the ML Framework

CHEFSI

```
bfs1 = np.array(...)
bfs2 = np.array(...)
w = np.array(...)
out = np.einsum("ik, jk, k->ij", bfs1, bfs2, w)
```

Array-ish (jax/pytorch/triton/helion) code

???

Tu, Jiqun, et al. ‘Accelerating High-Order Finite Element Simulations at Extreme Scale with FP64 Tensor Cores’. *arXiv Preprint arXiv:2603. 09038*, 2026.

Cui, Cu. ‘Acceleration of Tensor-Product Operations with Tensor Cores’. *ACM Transactions on Parallel Computing*, vol. 11, no. 4, ACM New York, NY, 2024, pp. 1–24.

Kusakabe, Ryota, et al. ‘GPU-Accelerated Sparse Matrix Vector Product Based on Element-by-Element Method for Unstructured FEM Using OpenACC’. *2022 Workshop on Accelerator Programming Using Directives (WACCPD)*, IEEE, 2022, pp. 52–61.

Collin, Teodoro Fields, et al. ‘Extensible Programmatic Specification of Finite Element Methods’. Under Submission To *ACM Transactions on Graphics*, 2025.

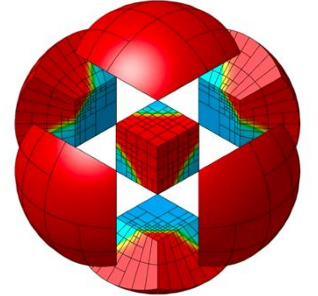

```
from EF import *

# Load a mesh.
mesh = TriangleVertexListReader(filename)

@pointwiseIntegral
def laplace(u: field(R), v: field(R)):
    return grad(u).dot(grad(v))

# Make a sparse matrix, the cotan Laplacian.
laplaceMatrix = assemble(mesh,[laplace],[LinearLagrange,LinearLagrange])

# Create Bounday Dofs
# on the boundary, what Dofs in the input spaces are relevant:
boundaryDofs = propagate(mesh.topologicalBoundary(),
                         laplaceMatrix.inputSections[0])

# setup boundary values
f = lambda x: ...
boundaryValues = assemble(mesh, LinearLagrange, [f])

# Setup linear system via elimination method for bcs
Ainterior = laplaceMatrix.reduce(["inputDofs0", "inputDofs1"]
                                ).getSlice([~boundaryDofs, ~boundaryDofs]
                                ).asScipyMatrix()
Abdry = laplaceMatrix.reduce(["inputDofs0", "inputDofs1"]
                            ).getSlice([~boundaryDofs, boundaryDofs]
                            ).asScipyMatrix()
b = boundaryValues.reduce(["outputDofs"]
                        ).getSlice(boundaryDofs
                        ).asDenseVector()
# Solve
uvals = spsolve(Ainterior, -Abdry.dot(b))
```

CHEFSI

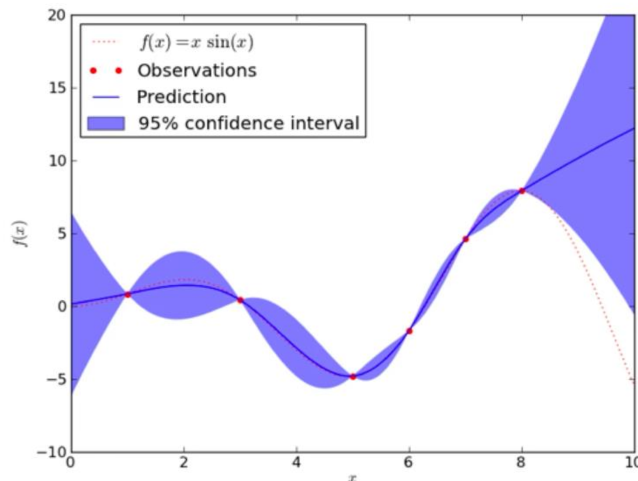
Differentiation: Connecting Science and AI

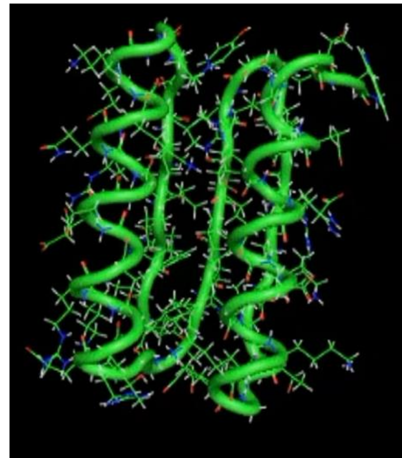
Derivatives are key to both scientific computing and machine learning

- Scientific Computing: UQ, Differential Equation, Error Analysis
- Machine Learning: Back-Propagation, Bayesian Inference



Links Science+AI: Compute the derivative of an application to leverage surrogate models, calibrate to observations, predict optimal forecast



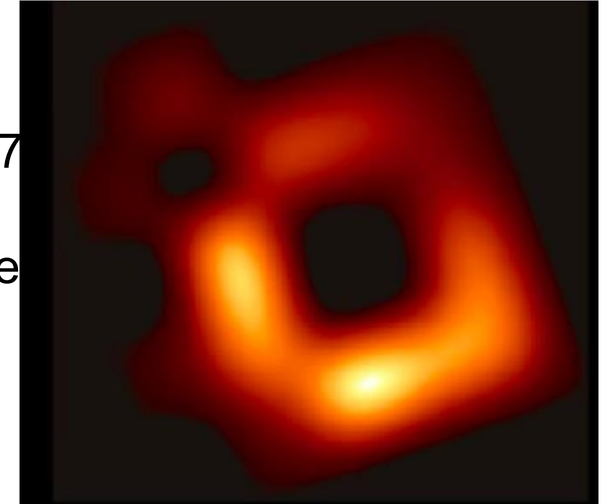


CHEFSI

Differentiation is Expensive

In AI and science, derivative computations are the most costly and difficult to use algorithms

Reconstructed image of M87
~1 week on cluster
Majority runtime is derivative

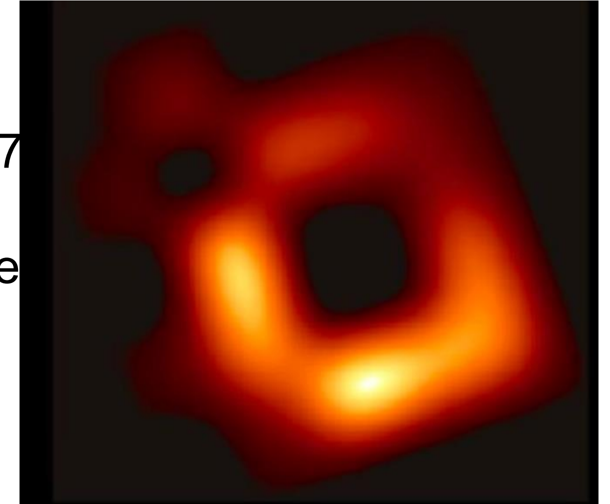


CHEFSI

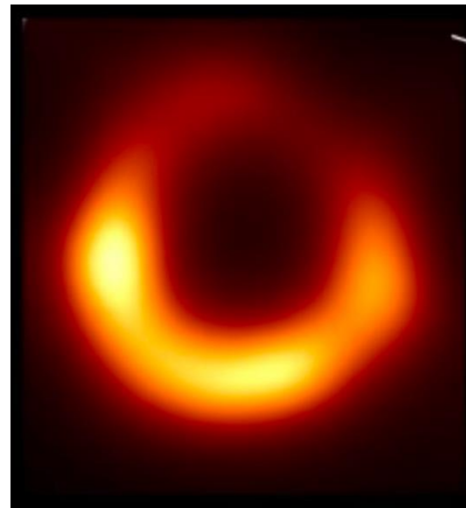
Differentiation is Expensive

In AI and science, derivative computations are the most costly and difficult to use algorithms

Reconstructed image of M87
~1 week on cluster
Majority runtime is derivative



With Enzyme differentiation:
1 hour on 1 thread

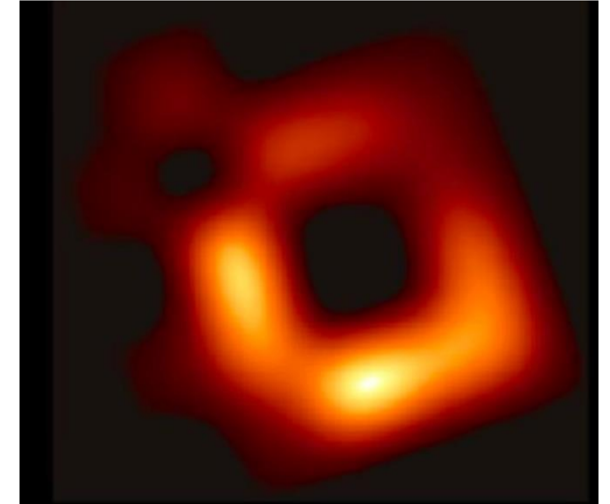


CHEFSI

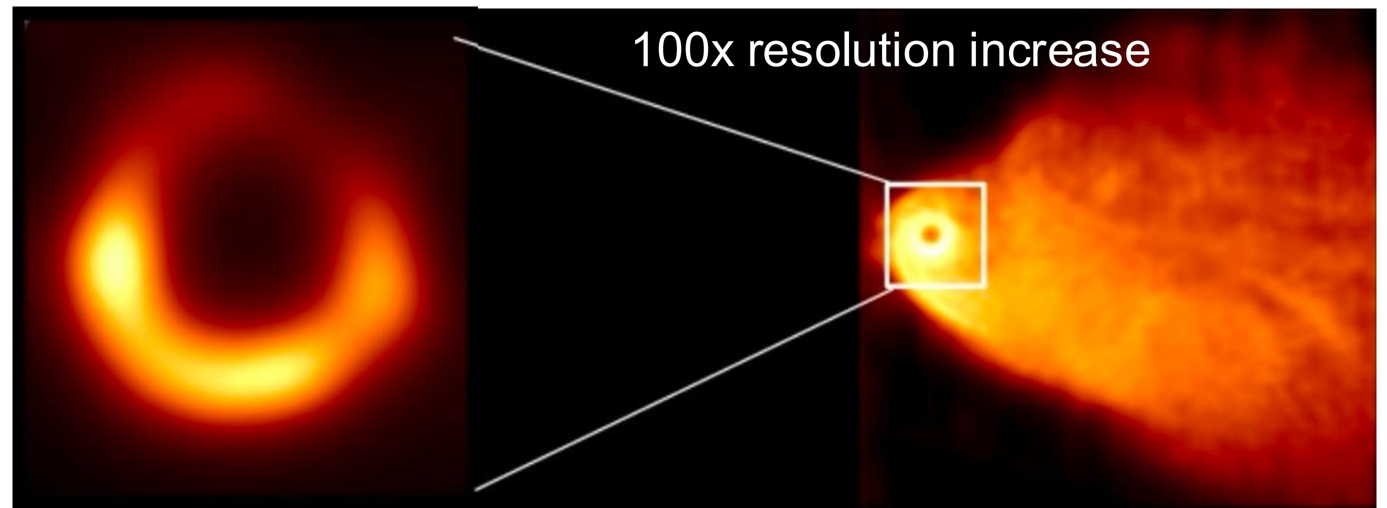
Differentiation is Expensive

In AI and science, derivative computations are the most costly and difficult to use algorithms

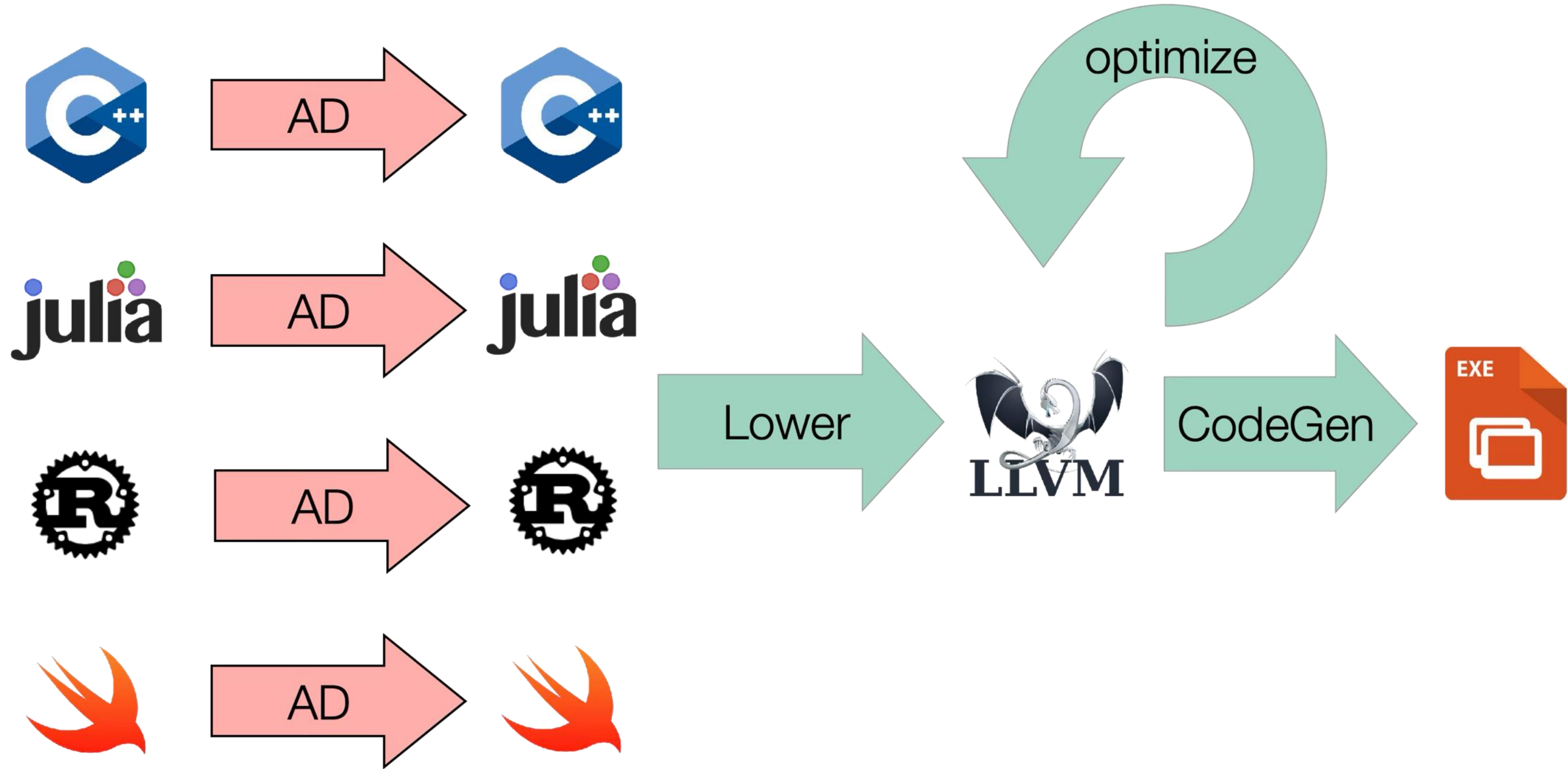
Reconstructed image of M87
~1 week on cluster
Majority runtime is derivative



With Enzyme differentiation:
1 hour on 1 thread



Existing Automatic Differentiation Pipelines



Enzyme Approach: compiler-based differentiation

AD within the compiler lets us work on **optimized** code!

optimize

optimize

Lower

LLVM

Enzyme

LLVM

CodeGen

EXE

Case Study: EnzymeAD + ΣMIT Integration

- Enzyme can successfully differentiate kernels in ΣMIT
- For gent-compressible test (min of 4 runs))
  - Tangent Speedup – 11.1x
  - Stress Speedup – 0.76x

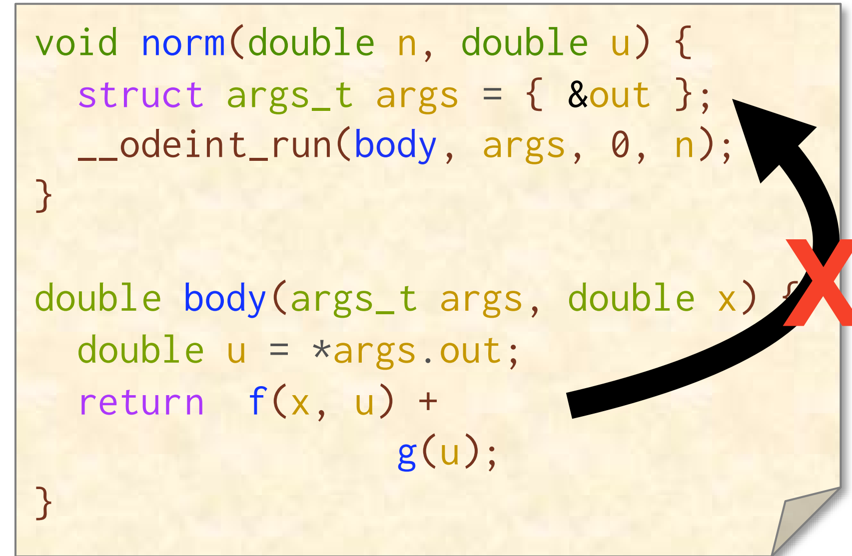
	Tangent (us)			Stress (us)		
	Analytical	Numerical	Enzyme	Analytical	Numerical	Enzyme
Min	20.100	4.490	1.810	0.190	2.020	0.250
Trial 1	20.260	4.620	2.010	0.190	2.020	0.260
Trial 2	20.670	4.520	1.890	0.220	2.089	0.260
Trial 3	23.730	4.980	1.840	0.220	2.060	0.290
Trial 4	20.100	4.490	1.810	0.220	2.090	0.250

```
// Within summit::MaterialConsistencyTest<MaterialType>::TestStress

std::vector<real> dq(nInt, 0.0);
dq[_potentialIndex] = 1.0;

__enzyme_autodiff<void>((void*)wrap_Constitutive<MaterialType>,
                        enzyme_const, &_material,
                        enzyme_const, F0,
                        enzyme_dup, F, P_enz,
                        enzyme_const, P,
                        enzyme_dupnoneed, nullptr, dq.data());
```

High-Level Structure of Scientific Code

- Multi-Physics code has structure that would be amenable to optimization
- Mainstream compilers cannot perform these optimizations
- High-level optimizations are critical to derivative (and primal) performance

$$\int_0^N f(x,u) + g(u)\, dx \quad \longrightarrow \quad Ng(u) + \int_0^N f(x,u)dx$$



$$\frac{\partial}{\partial x_i}\left(\frac{\partial f}{\partial x_j}\right) = \frac{\partial}{\partial x_j}\left(\frac{\partial f}{\partial x_i}\right)$$

Summary: Challenges of Scaling MultiPhysics

- How do we speed up **solvers**?
- How do we take advantage of (distributed) **GPUs**?
- How can FEM software leverage **ML infrastructure**?
- How do we improve **derivatives** and **floating-point performance**?
- How do we apply new insights to **legacy** software?

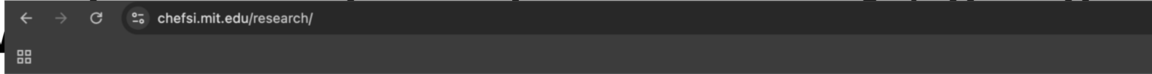
Summary: Challenges of Scaling MultiPhysics

- How do we speed up **solvers**?
- How do we take advantage of (distributed) **GPUs**?
- How can FEM software leverage **ML infrastructure**?
- How do we improve **derivatives** and **floating-point performance**?
- How do we apply new insights to **legacy** software?

- Automation through compiler technology
- Retain high-level properties throughout the software stack
- Co-design between computer scientists and physical scientists!

Summary: Challenges of Scaling MultiPhysics

- How do we speed up **solvers**?
- Hov ted) **GP**
- Hov **ıfrastru**
- Hov **loating-**
- How do we apply new insights to **legacy** softw



MIT Center for the Exascale Simulation of Coupled High Enthalpy Fluid-Solid Interactions

A US Department of Energy Predictive Science Center Sponsored by the National Nuclear Security Administration

- Automation through compiler technology
- Retain high-level properties throughout the software stack
- Co-design between computer scientists and physical scientists!