Introduction to Numerical Software

Presented to
ATPESC 2026 Participants

Carol S. Woodward
Lawrence Livermore National Laboratory

Date 07/29/2026

ATPESC Numerical Software Track

Agenda

Time	Amphitheater	Breakout
1:30 – 1:50	Introduction to Numerical Software – Carol Woodward	
1:50 – 3:10	Structured Discretizations with AMReX – Andrew Myers, Weiqun Zhang	Unstructured Discretizations with MFEM/PUMI – Mark Shephard
3:10 – 3:40	Break	
3:40 – 5:00	Nonlinear Solvers with PETSc – Richard Mills	Time Integrators with SUNDIALS – Dan Reynolds
5:00 – 6:20	Sparse Direct Linear Solvers with SuperLU and STRUMPACK – Yang Liu	Iterative Linear Solves and Preconditioners with hypre - Victor Magri and Daniel Osei-Kuffuor
6:20 – 6:30	Wrap-up – Carol Woodward	
6:30 – 7:30	Dinner	
	Thursday Morning	
???	Hands-on Exercises	

Your home bases for the day: ATPESC Track 4a
Numerical Algorithms and Software for Extreme-Scale Science

- Main ATPESC Agenda
  - https://extremecomputingtraining.anl.gov/2026-agenda

- Math Packages Training Site
  - session abstracts, links to parallel breakout rooms, hands-on lessons, more
  - https://xsdk-project.github.io/MathPackagesTraining2026/

Using Slack

Recommend using the desktop app, but browser ok too

- **# atpesc-2026-track-4a-numerical-sw** channel
  - For all chat during presentations
  - For all chat outside any specific parallel session
  - Recommend using threads to help keep conversations specific

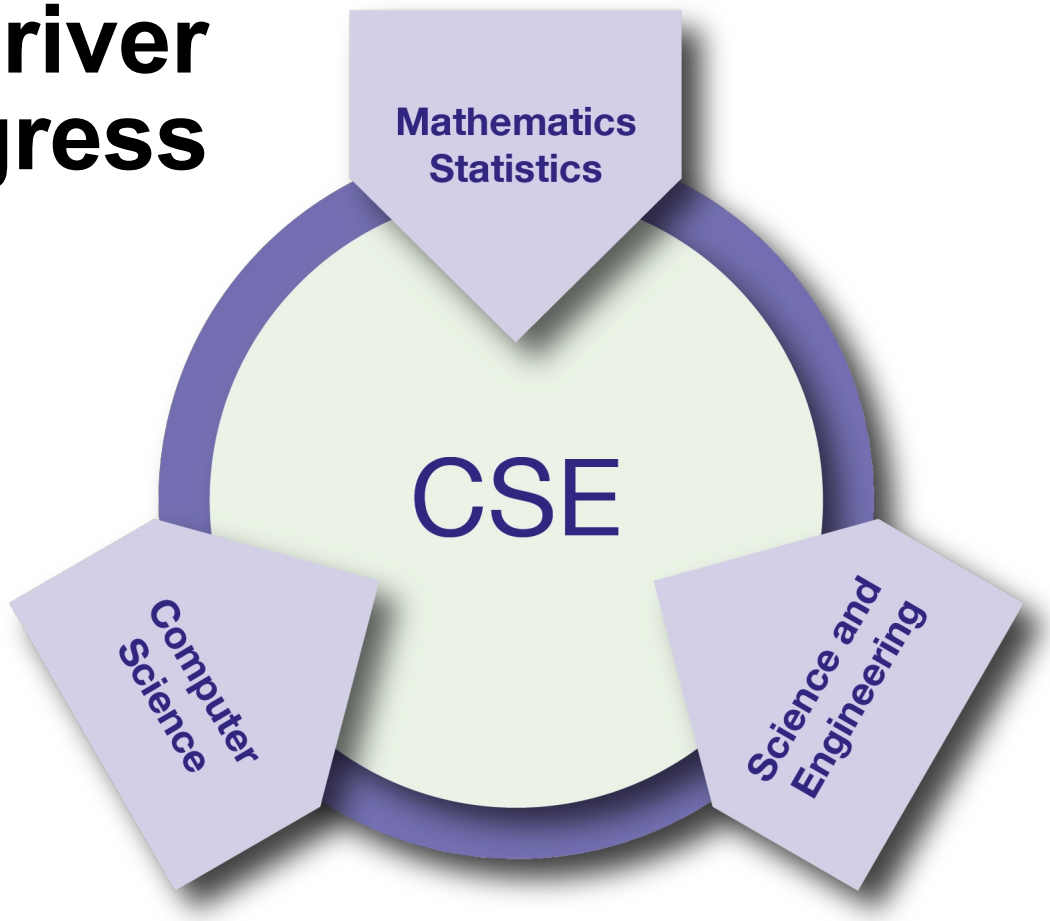
- **# atpesc-2026-helpdesk** channel
  - For general help

CSE: Essential driver of scientific progress

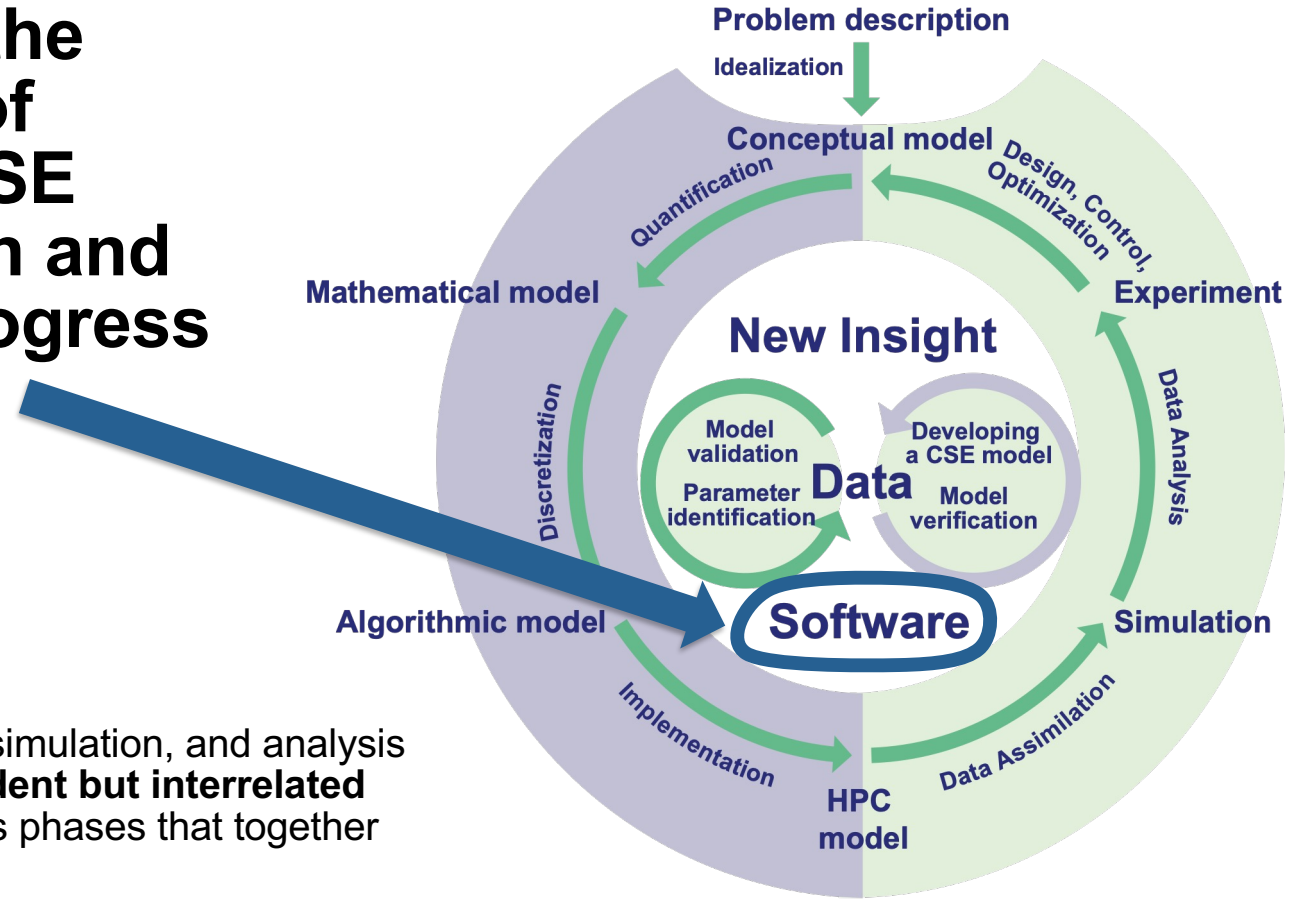
CSE = Computational Science & Engineering

Development and use of computational methods for scientific discovery

- all branches of the sciences
- engineering and technology
- support of decision-making across a spectrum of societally important applications



Software is the foundation of sustained CSE collaboration and scientific progress

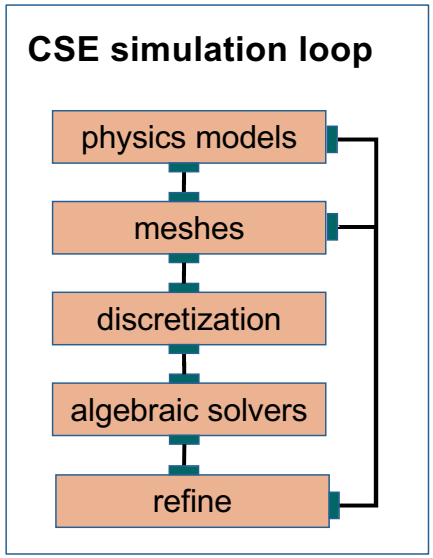


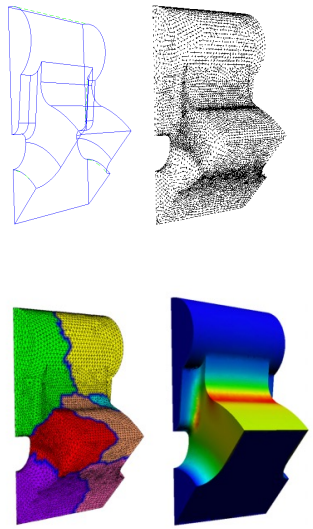
CSE cycle: Modeling, simulation, and analysis

- **Software: independent but interrelated elements** for various phases that together enable CSE

Computational Science and Engineering (CSE) simulation starts with a forward simulation that captures the physical phenomenon of interest

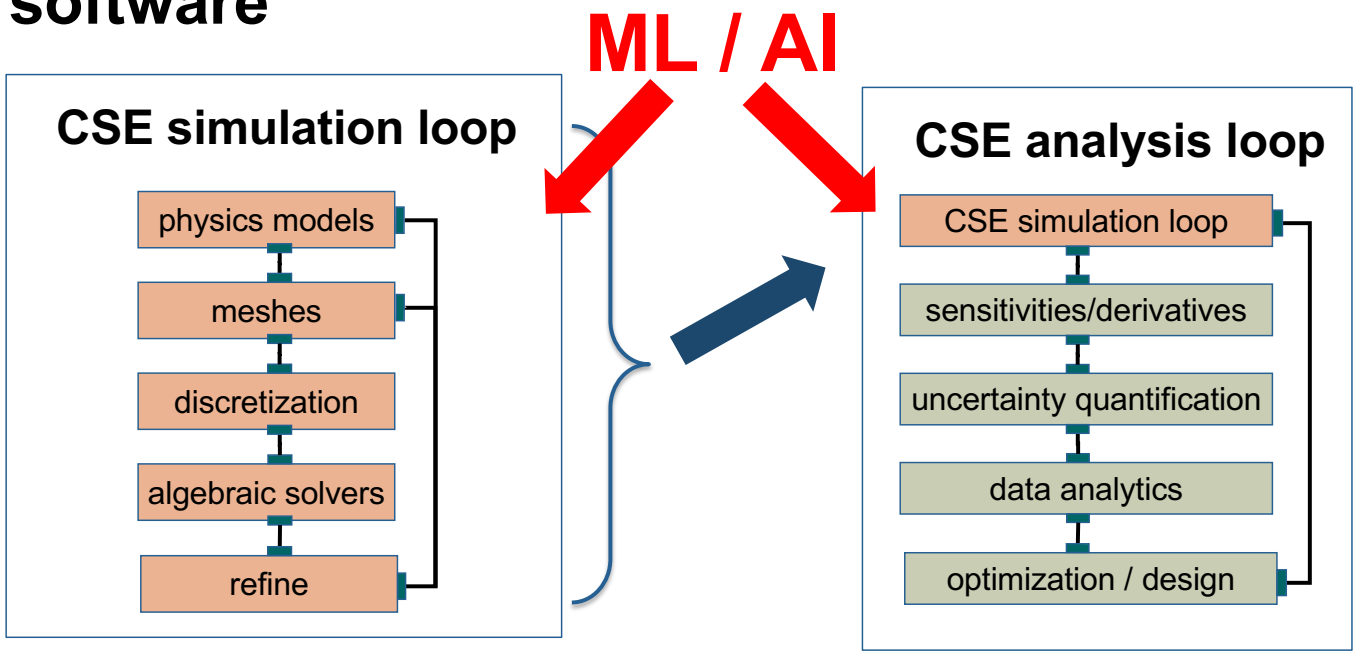
- Develop a mathematical model of the phenomenon of interest
- Approximate the model using a discrete representation
- Solve the discrete representation
- Adapt and refine the mesh or model
- Incorporate different physics, scales





Requires: mesh generation, partitioning, load balancing, high-order discretization, time integration, linear & nonlinear solvers, eigensolvers, mesh refinement, multiscale/multiphysics coupling, etc.

Decision support methods build on the CSE simulation loop … and rely on even more numerical algorithms and software



Beyond interpretive simulations …

working toward predictive science

Requires: adjoints, sensitivities, algorithmic differentiation, sampling, ensembles, data analytics, uncertainty quantification, optimization (derivative free & derivative based), inverse problems, etc.

First consider a very simple example

- 1D rod with one end in a hot water bath, the other in a cold water bath
- Mathematical model

$$\nabla^2 T = 0 \in \Omega$$

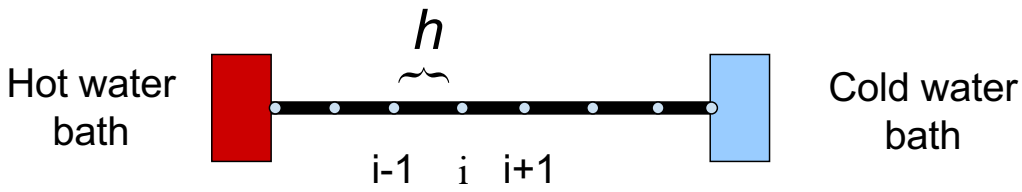
$$T(0) = 180^o \quad T(1) = 0^o$$

The first step is to discretize the equations

- Approximate the derivatives of the continuous equations with a discrete representation that is easier to solve
- One approach: Finite differences

$$\nabla^2 T \approx (T_{i+1} - 2T_i + T_{i-1})/h^2 = 0$$

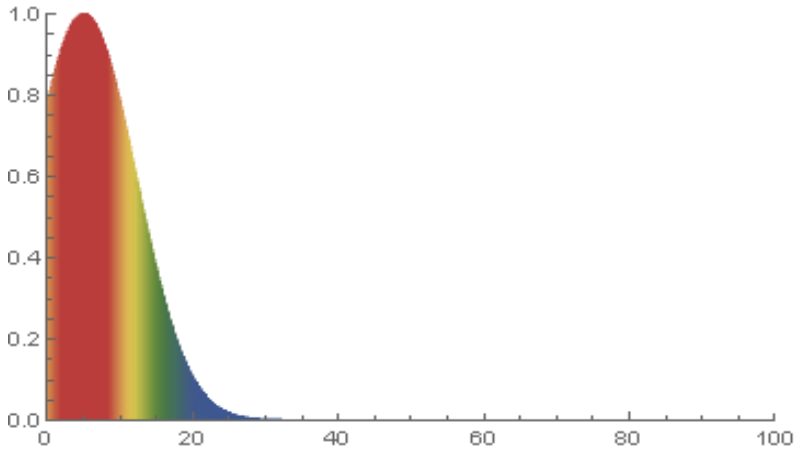
$$T_0 = 180^o \qquad T_n = 0^o$$



Then you can solve for the unknowns T_i

- Set up a matrix of the unknown coefficients
  - include the known boundary conditions
- Solve the linear system for T_i

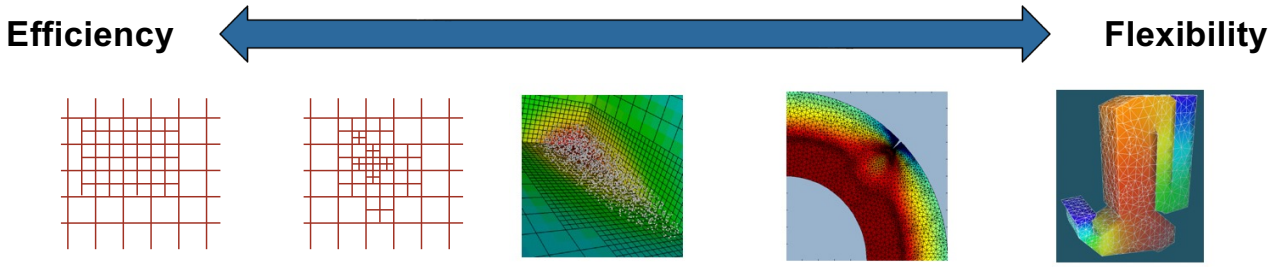
$$\begin{pmatrix} 2 & -1 & 0 & \ldots\ldots\ldots\ldots & 0 \\ -1 & 2 & -1 & 0 & \ldots\ldots\ldots & 0 \\ 0 & -1 & 2 & -1 & 0 \ldots\ldots & 0 \\ & & \ldots\ldots\ldots & & & \\ 0 & \ldots\ldots\ldots\ldots & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} T_1 \\ T_2 \\ T_3 \\ . \\ T_{n-1} \end{pmatrix} = \begin{pmatrix} 180\, h^2 \\ 0 \\ 0 \\ . \\ 0 \end{pmatrix}$$



- Visualize and analyze the results

As problems get more complicated, so do the steps in the process

- Different discretization strategies exist for differing needs



- Most problems are time dependent and nonlinear
  - Need higher algorithmic levels than linear solvers
  - Nonlinear solvers, time integrators
- Increasingly combining multiple physical processes (e.g., rod with water bath combined with an outside stress)
  - Interactions require careful handling
- Goal-oriented problem solving requires optimization, uncertainty quantification

Today, you will hear about and work with several packages from the FASTMath Institute

- FASTMath brings together a large team and stewards an extensive library of highly used performant software for the Dept. of Energy
- The team consists of ~60 researchers from 6 national laboratories and 4 universities
- Development team has decades of numerical software experience
- Most packages run on laptops to exascale systems
- Packages support discretizations, algebraic solvers, and decision support methods
- Most are adopting machine learning accelerations within their algorithms

FASTMath development teams are looking toward the future of scientific computing with AI

- New AI coding agents are turning code developers into *code developer managers*
- Vibe coding - Agents do the coding, and scientists direct the agents toward science application goals
- What is the role of numerical libraries in this future?
  - Informing AI engines as to which methods fit a given situation
  - Providing new and advanced algorithms
  - **Preventing the AIs from delivering inefficient first order methods**

Vibe Coding a FASTMath Software Demonstration with OpenAI's ChatGPT 5.4 and Codex

Cody Balos on the SUNDIALS team tested out vibe coding an AMReX-SUNDIALS demo…

Initial Prompt:

> *Please build a program that solves a scalar-valued advection-diffusion problem for chemical transport using AMReX and SUNDIALS. Use Spack for installing any dependencies and for building the code.*

```
• Plan:

  1. in_progress Scaffold the project files: source, input deck, CMake, Spack environment, and a local Spack package.
  2. pending Build dependencies and the application through Spack, then fix any API or packaging issues.
  3. pending Run a short verification case and tighten the README with exact build/run commands.

• Updated Plan
  └ The repository is empty, so I'm creating a complete minimal application and making Spack responsible for both dependency resolution and building
    the project itself.
    □ Scaffold the project files: source, input deck, CMake, Spack environment, and a local Spack package.
    □ Build dependencies and the application through Spack, then fix any API or packaging issues.
    □ Run a short verification case and tighten the README with exact build/run commands.

• I have the dependency versions and variant surface now. I'm narrowing this to a CPU-only, MPI-enabled 2D example with AMReX for the mesh/data path
  and SUNDIALS ARKODE for time integration, which keeps the build tractable while still matching the request.
```

~20 minutes and 400k tokens later...

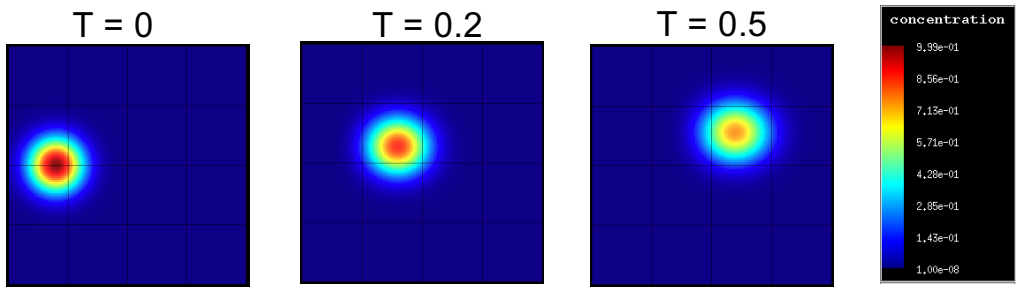
```
• Built a working AMReX + SUNDIALS scalar transport application and wired it to Spack end to end. The solver lives in src/main.cpp, uses AMReX for mesh/MultiFab/plotfiles, and uses SUNDIALS ARKODE with the parallel N_Vector for time
  integration of an advection-diffusion concentration field. The project is built through the local Spack package in spack-repo/packages/chem-transport/package.py, with the environment in spack.yaml and a pinned concretization in
  spack.lock.

  The default case and run instructions are in inputs/chem_transport.inputs and README.md. I verified spack -e . install succeeds, then ran:

  - mpiexec -n 1 ... transport.final_time=0.02 ..., which completed at t = 0.02 in 20 internal steps.
  - mpiexec -n 2 ... transport.final_time=0.01 ..., which completed at t = 0.01 in 12 internal steps.

  The repository is now ready to use from the root with the documented Spack commands.
```

- In one-shot, Codex produced a working, but not perfect, MPI-enabled parallel demonstration.
- It chose to use a SUNDIALS-provided data structure (stores the problem state) and copy to/from the AMReX data structure... this is suboptimal
- With a follow-up prompt we tell it to use the SUNDIALS-wrapped AMReX structure that is available in AMReX, to get rid of the extra copies (it lets SUNDIALS operate directly on the AMReX data structure)
- It did not provide any visualization, so with a follow-up prompt we ask it to enable visualizing with the AMReX Amrvis tool

A reasonable working demonstration problem was achieved in less than an hour

- Spent ~5 minutes going back and forth, prompting and examining results, to refine the demo

Plot of particle concentration evolved from t=0 to t=0.5 on a 128^2 mesh with 2 MPI ranks



- A more specific initial prompt can produce a much better result in one-shot, but for new users it may be difficult to be specific right away

Check out the demo on GitHub

https://github.com/sundials-codes/codex-demonstration

The code solves

$$\frac{\partial c}{\partial t} + \nabla \cdot (u c) = D \nabla^2 c$$

for a scalar concentration field $c(x, y, t)$ on a uniform Cartesian mesh. In the current implementation:

- the code is strictly 2D (`AMREX_SPACEDIM == 2`)
- the domain is rectangular and periodic in both directions
- the velocity $u = (u_x, u_y)$ is spatially constant
- the diffusion coefficient D is a single constant scalar
- the initial condition is a Gaussian pulse plus a uniform background level

The default initial state is

$$c(x, y, 0) = c_{\text{background}} + A \exp\left(-\frac{1}{2} \left[\left(\frac{x - x_0}{w_x} \right)^2 + \left(\frac{y - y_0}{w_y} \right)^2 \right] \right)$$

where A is `transport.pulse_amplitude` , (x_0, y_0) is `transport.pulse_center` , and (w_x, w_y) is `transport.pulse_width` .

Input Options

These are the application-specific ParmParse options currently read by the executable. Array-valued entries take two numbers because the code is 2D.

Option	Default	Description
`transport.n_cell`	`128 128`	Number of cells in `x` and `y` .
`transport.max_grid_size`	`32`	Maximum AMReX box size used when chopping the domain for parallel distribution.
`transport.prob_lo`	`0.0 0.0`	Physical lower corner of the rectangular domain.
`transport.prob_hi`	`1.0 1.0`	Physical upper corner of the rectangular domain.
`transport.velocity`	`1.0 0.35`	Constant advection velocity `(u_x, u_y)` .
`transport.diffusion`	`2.5e-4`	Constant isotropic diffusion coefficient `D` .
`transport.background`	`1.0e-8`	Uniform concentration offset added everywhere at initialization.
`transport.pulse_amplitude`	`1.0`	Peak amplitude of the Gaussian pulse above the background.
`transport.pulse_center`	`0.2 0.5`	Center of the Gaussian pulse in physical coordinates.
`transport.pulse_width`	`0.08 0.08`	Gaussian width parameters in each direction.
`transport.final_time`	`0.5`	Final simulation time passed to ARKODE.
`transport.output_interval`	`0.1`	Time between plotfiles. If this is `<= 0` , only the initial and final states are written.
`transport.rtol`	`1.0e-6`	Relative tolerance used by `ARKodeSStolerances` .
`transport.atol`	`1.0e-10`	Absolute tolerance used by `ARKodeSStolerances` .
`transport.init_step`	`1.0e-4`	Initial internal time step suggested to ARKODE.
`transport.max_step`	`5.0e-3`	Upper bound on the internal time step size.
`plot.prefix`	`plt`	Prefix for AMReX plotfiles, producing names like `plt00000` .

Conclusions

- Software packages with robust implementations of highly efficient algorithms are available for CSE on high performance computing platforms
- Vibe coders need to understand
  - Method tradeoffs and why some methods would be better than others
  - Significant inefficiencies can arise with vibe coding and an understanding of package capabilities (and limitations) is needed
- Upcoming features:
  - More AGENTS.md and SKILLS.md files to help agents know which methods to use and how to use them
  - Token-minimizing interfaces
  - Mixed precision algorithms throughout the software stack
  - Machine learning used to make packages “smart” and/or to accelerate packages

Block-structured adaptive mesh refinement framework. Scalable support for hierarchical mesh and particle data, with embedded boundaries.

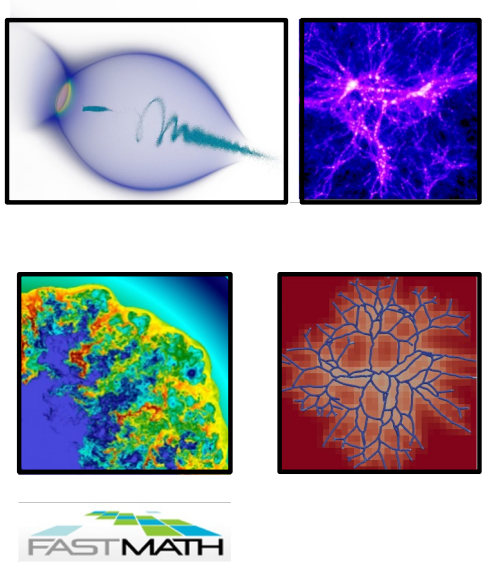
Capabilities

- Support for PDEs on a hierarchical adaptive mesh with particles and embedded boundary representations of complex geometry
- Support for multiple modes of time integration
- Support for explicit and implicit single-level and multilevel mesh operations, multilevel synchronization, particle, particle-mesh and particle-particle operations
- Hierarchical parallelism –
  - hybrid MPI + OpenMP with logical tiling on multicore architectures
  - hybrid MPI + GPU support for hybrid CPU/GPU systems (NVIDIA CUDA, AMD HIP, Intel SYCL)
- Native multilevel geometric multigrid solvers for cell-centered and nodal data
- Highly efficient parallel I/O for checkpoint/restart and for visualization – native format supported by Visit, Paraview, yt

Open source software

- Used for diverse apps, including accelerator modeling, astrophysics, combustion, cosmology, multiphase flow, phase field modeling, atmospheric modeling and more
- Source code and development hosted on github with rigorous testing framework
- Extensive documentation, examples and tutorials

Examples of AMReX applications



https://www.github.com/AMReX-Codes/amrex

MFEM

Lawrence Livermore National Laboratory

Free, lightweight, scalable C++ library for finite element methods. Supports arbitrary high order discretizations and meshes for wide variety of applications.

Flexible discretizations on unstructured grids

- Triangular, quadrilateral, tetrahedral and hexahedral meshes.
- Local conforming and non-conforming refinement.
- Bilinear/linear forms for variety of methods: Galerkin, DG, DPG, …

High-order and scalable

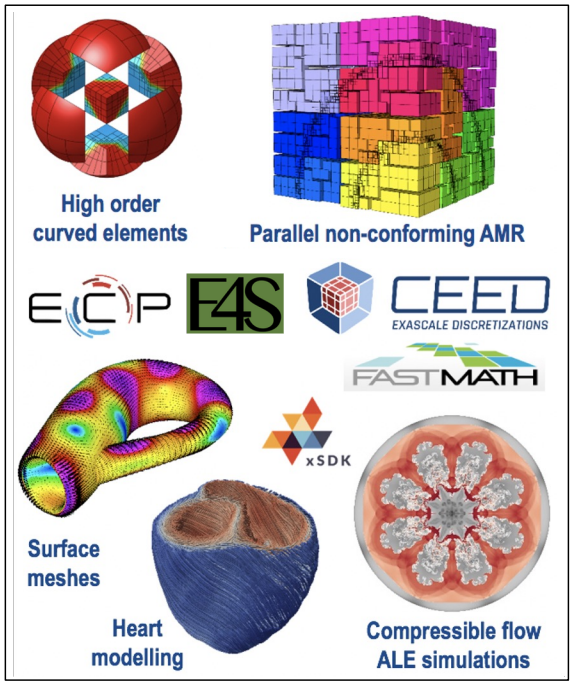
- Arbitrary-order H1, H(curl), H(div)- and L2 elements. Arbitrary order curvilinear meshes.
- MPI scalable to millions of cores and includes initial GPU implementation. Enables application development on wide variety of platforms: from laptops to exascale machines.

Built-in solvers and visualization

- Integrated with: HYPRE, SUNDIALS, PETSc, SUPERLU, …
- Accurate and flexible visualization with VisIt and GLVis

Open source software

- BSD with thousands of downloads/year worldwide.
- Available on GitHub, also via OpenHPC, Spack.



https://mfem.org

Parallel Unstructured Mesh Interface (PUMI)

Parallel management and adaptation of unstructured meshes.
Interoperable components to support the development of unstructured mesh simulation workflows

PUMI General and Portable Adapt

- Distributed, conformant mesh with entity migration, remote read only copies, fields and their operations
- Mesh adaptation (straight and curved), mesh motion

PUMI Particle

- GPU accelerated particle operations over unstructured meshes.
- Distributed memory parallelism supported via particle migration and load balancing procedures.

SCOREC

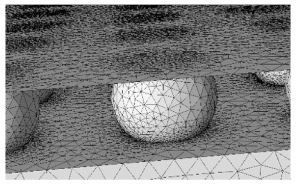
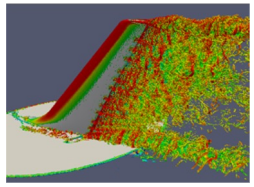
PCMS

- Intrinsic and extrinsic code coupling library for exascale applications.
- Field mapping and transfer functions supporting multiple mesh, grid, and coordinate systems.

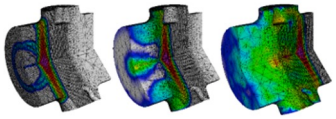
Rensselaer

Supported Applications

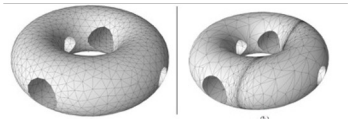
- MFEM: High order FE framework
- PetraM: Adaptive RF fusion
- PHASTA: FE for turbulent flows
- FUN3D: FV CFD
- Proteus: Multiphase FE
- ACE3P: High order FE for EM
- M3D-C1: FE based MHD
- Nektar++: High order FE for flow
- Albany/Trilinos: Multi-physics FE
- GITRm: impurity transport
- XGCm: core+edge fusion plasma physics
- PolyMPO: GPU polygonal mesh-based material point operations
- PUMI-Tally: supports Monte Carlo neutral particle transport

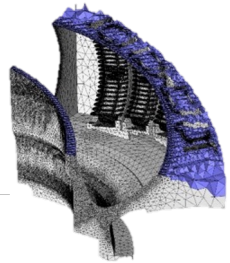
Applications with billions of elements: flip-chip (L), flow control (R)



Mesh adaptation for evolving features



Anisotropic adaptation for curved meshes



RF antenna and plasma surface in vessel.

More Info: scorec.rpi.edu/software
CPU Mesh Adapt Paper: doi.org/10.1145/2814935
GPU Mesh Adapt Thesis: hdl.handle.net/20.500.13015/1817

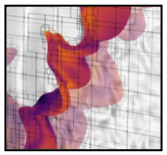
SUNDIALS

Suite of Nonlinear and Differential /Algebraic Equation Solvers

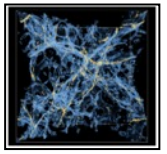
Adaptive time integrators for ODEs and DAEs and efficient nonlinear solvers
Used in a variety of applications. Freely available. Encapsulated solvers & parallelism.

- **ODE and DAE time integrators:**
  - *CVODE:* adaptive order and step BDF (stiff) & Adams (non-stiff) methods for ODEs
  - *ARKODE:* adaptive step implicit, explicit, IMEX, and multirate Runge-Kutta methods for ODEs
  - *IDA*: adaptive order and step BDF methods for DAEs
  - *CVODES* and *IDAS:* provide forward and adjoint sensitivity analysis capabilities
- **Nonlinear Solvers:** *KINSOL* – Newton-Krylov; accelerated Picard and fixed point
- **Modular Design:** Easily incorporated into existing codes; Users can supply their own data structures and solvers or use SUNDIALS provided modules
- **Support on NVIDIA, AMD, and Intel GPUs:**
  - Vectors: CUDA, HIP, OpenMP Offload, RAJA, SYCL (DPC++)
  - Linear solvers: cuSOLVER, MAGMA, matrix-free Krylov methods
- **Open Source:** BSD License; Download from LLNL site, GitHub, or Spack
  - Supported by extensive documentation; user email list with an active community
  - Available through MFEM, AMReX, deal.II, and PETSc

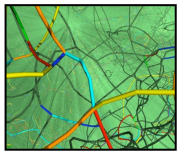
SUNDIALS is used worldwide in applications throughout research and industry



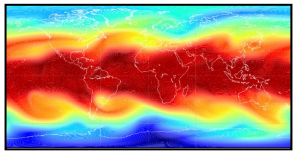
Combustion (Pele)



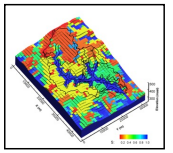
Cosmology (Nyx)



Dislocation dynamics (ParaDiS)



Atmospheric Dynamics (Tempest)



Subsurface flow (ParFlow)

FASTMATH

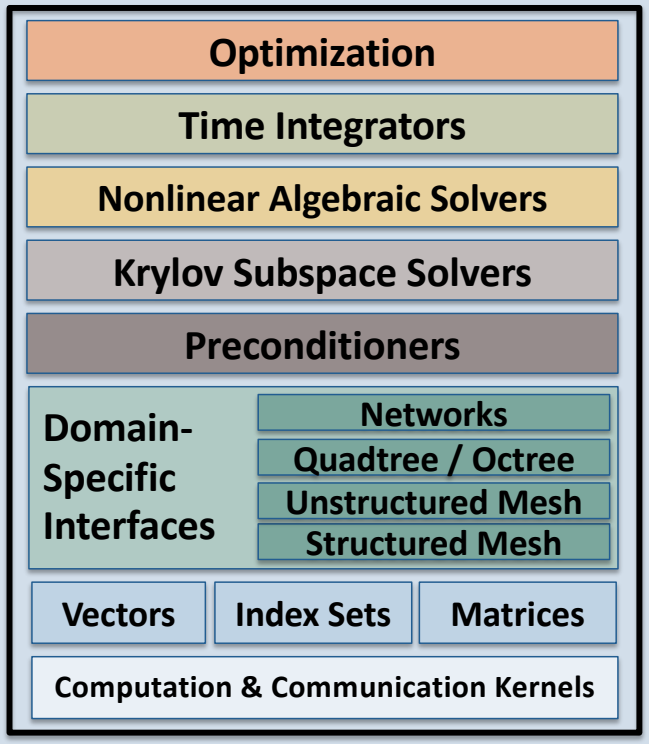
xSDK

http://www.llnl.gov/casc/sundials

PETSc TAO

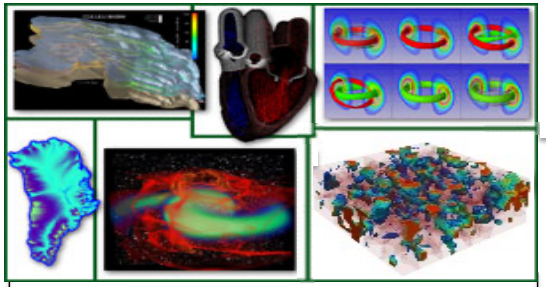
Portable, Extensible Toolkit for Scientific Computation / Toolkit for Advanced

Scalable algebraic solvers for PDEs. Encapsulate parallelism in high-level objects. Active & supported user community. Full API from Fortran, C/C++, Python.



- **Easy customization and composability of solvers at runtime**
  - Enables optimality via flexible combinations of physics, algorithmics, architectures
  - Try new algorithms by composing new/existing algorithms (multilevel, domain decomposition, splitting, etc.)
- **Portability & performance**
  - Largest DOE machines, also clusters, laptops; NVIDIA, AMD, and Intel GPUs
  - Thousands of users worldwide

Argonne NATIONAL LABORATORY



PETSc provides the backbone of diverse scientific applications. clockwise from upper left: hydrology, cardiology, fusion, multiphase steel, relativistic matter, ice sheet modeling

NUMFOCUS OPEN CODE = BETTER SCIENCE E4S FASTMATH R&D 100

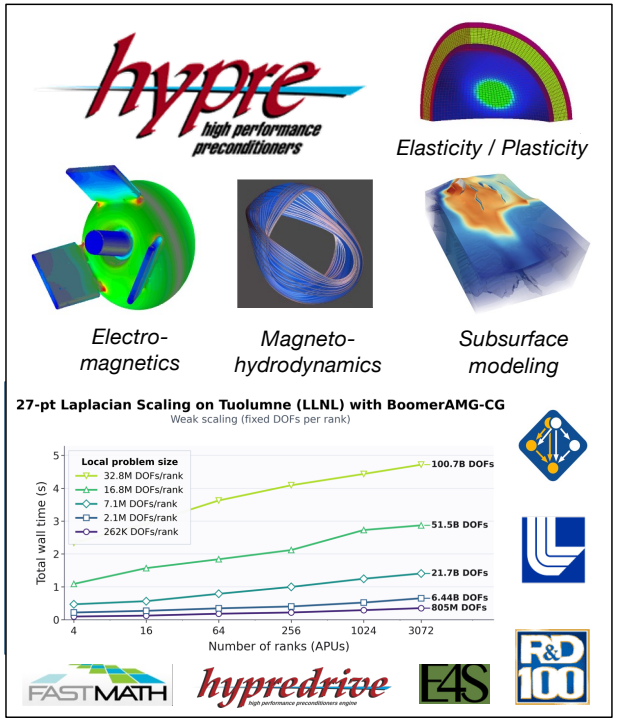
https://petsc.org

hypre

Lawrence Livermore National Laboratory

Scalable multilevel linear solvers & preconditioners. Unique user-friendly interfaces. Flexible software design. Used in a vast array of applications. Freely available.

- **Conceptual interfaces**
  - Structured, semi-structured, and linear-algebraic interfaces
  - Provide natural "views" of the linear system
  - Enable efficient (scalable) linear solvers via effective data storage schemes
- **Scalable preconditioners and iterative solvers**
  - (Semi-)Structured (SSAMG/PFMG/SMG) and algebraic multigrid (BoomerAMG)
  - Multigrid-based auxiliary space solvers for H-curl (AMS) and H-div (ADS)
  - Multigrid solvers for nonsymmetric systems (pAIR)
  - Multigrid reduction for systems of PDEs (MGR)
  - Incomplete and approximate factorization (ILU/FSAI) preconditioners
  - Matrix-free/mixed-precision Krylov solvers and multi-precision library support
- **Exascale systems support and GPU-readiness via different backends**
  - Nvidia GPUs (CUDA), AMD GPUs (HIP), and Intel GPUs (SYCL)
- **Open-source software**
  - Can be used through other math libraries, e.g., PETSc and Trilinos
  - Available on GitHub: https://www.github.com/hypre-space/hypre
  - Flexible control via *hypredrive*: https://www.github.com/hypre-space/hypredrive



http://www.llnl.gov/CASC/hypre

SuperLU

Supernodal Sparse LU Direct Solver. Flexible, user-friendly interfaces. Examples show various use scenarios. Testing code for unit-test. BSD license.

Capabilities

- Serial (thread-safe), shared-memory (SuperLU_MT, OpenMP or Pthreads), distributed-memory & multi-GPUs (SuperLU_DIST, hybrid MPI+ OpenMP + CUDA/HIP). Written in C/C++, with Fortran, Julia, Python interfaces
- Sparse LU decomposition (can be nonsymmetric sparsity pattern), symmetric LDL' decomposition, triangular solution with multiple right-hand sides
- Incomplete LU (ILUTP) preconditioner in serial SuperLU
- Sparsity-preserving ordering: minimum degree or graph partitioning applied to A^TA or A^T+A
- User-controllable pivoting: partial pivoting, threshold pivoting, static pivoting
- Condition number estimation, iterative refinement, componentwise error bounds
- Batch interface, mixed-precision routines

Exascale systems GPU-readiness

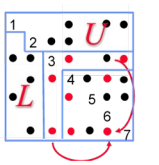
- Available: Nvidia GPU (CUDA), AMD GPU (HIP), Intel GPU (SYCL)

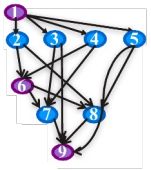
Parallel Scalability

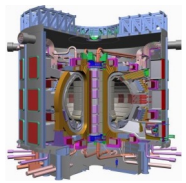
- Factorization strong scales to 32,000 CPU cores (IPDPS'18, JPDC'19)
  - additional 10x speedup with GPUs
- Triangular solve strong scales to 4000 CPU cores (SIAM CSC'18, SIAM PP'20, SC'23)
  - 3D algorithm strong scales to 256 GPUs

Open-source software

- BSD licensee; available on github
- Used in a vast range of applications, can be used through hypre, PETSc, SUNDIALS, Trilinos, etc.







ITER tokamak

quantum mechanics

Widely used in commercial software, including AMD (circuit simulation), Boeing (aircraft design), Chevron, ExxonMobile (geology), Cray's LibSci, FEMLAB, HP's MathLib, IMSL, NAG, SciPy, OptimaNumerics, Walt Disney Animation.

https://portal.nersc.gov/project/sparse/superlu/

STRUMPACK

STRUctured Matrix PACKkage

Hierarchical solvers for dense rank-structured matrices and fast algebraic sparse solver and robust and scalable preconditioners.

Dense Matrix Solvers using Hierarchical Approximations

- Hierarchical partitioning, low-rank approximations
- Hierarchically Semi-Separable (HSS), Hierarchically Off-Diagonal Low-Rank (HODLR), Hierarchically Off-Diagonal Butterfly (HODBF), Block Low-Rank (BLR), Butterfly
- C++ Interface to ButterflyPACK (Fortran)
- Applications: BEM, Cauchy, Toeplitz, kernel & covariance matrices, ...
- Asymptotic complexity much lower than LAPACK/ScaLAPACK routines

Sparse Direct Solver

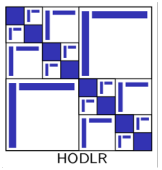
- Algebraic sparse direct solver
- GPU: CUDA, HIP/ROCm, DPC++ (in progress)
- Orderings: (Par)METIS, (PT)Scotch, RCM

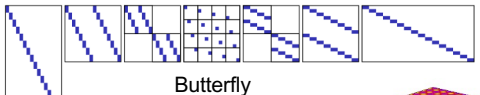
Preconditioners

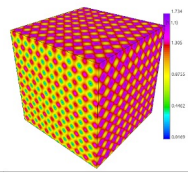
- Approximate sparse factorization, using hierarchical matrix approximations
- Scalable and robust, aimed at PDE discretizations, indefinite systems, …
- Iterative solvers: GMRES, BiCGStab, iterative refinement

Open-source Software

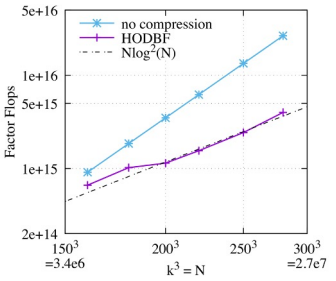
- BSD license; available on github
- Interfaces from PETSc, MFEM, Trilinos, available in Spack

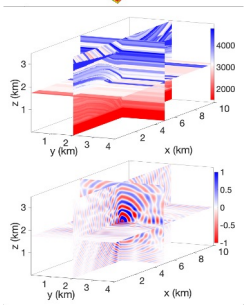






Near linear scaling for high-frequency wave equations





github.com/pghysels/STRUMPACK

CASC

Center for Applied Scientific Computing

Lawrence Livermore National Laboratory

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC. LLNL-PRES-852746

This work was supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research (ASCR), Scientific Discovery through Advanced Computing (SciDAC) program, and by the Exascale Computing Project, a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.