Exascale Predictive Simulation of Inductively Coupled Plasma Torches

Bob Moser and George Biros
Oden Institute for Computational Engineering and Sciences, UT-Austin
PSAAP3 Kickoff Meeting, 18 August 2020

PECOS
Predictive Engineering and Computational Sciences

https://pecos.oden.utexas.edu
Texas MSC Project Objectives

Perform predictive simulations of an inductively coupled plasma (ICP) torch

- Predict outlet flow characteristics and limits of stable operation
- Consider Argon and air feed gas, and varying flow rate and pressure
- Make validated predictions with quantified uncertainty
- Identify processes that limit stable operation
- Enable improved ICP torch design

Advance computational science capabilities

- Advance plasma physics modeling
- Advance exascale computational performance and productivity
- Advance exascale and UQ algorithms
- Advance predictive validation and UQ
- To the benefit of the NNSA labs

We have an ICP torch facility at the Pickle research campus
Inductively Coupled Plasma Torch

- Feed gas enters at bottom
- Power deposited into gas by RF induction coil
- Electromagnetic fields accelerate electrons, which heat and ionize through collisions
- Peak plasma temperature \(~10,000\text{K}\) with 1\% ionization
- The relatively high pressure outlet plasma can be used for various purposes
ICP Torch Characteristics

- High-speed tangential inlet jets introduce swirl
- Swirl stabilizes plasma by segregating low density gas toward the center
- Flow in the hot region is laminar, while inlet and exit jets are turbulent
- Thermo-chemical state of the outlet jet is of interest in applications
- Stable operation only possible over limited range of conditions (flow rate, feed gas, pressure) but governing mechanism not well understood
ICP Torch at UT Austin
ICP Torch Applications

**ICP torches have a wide variety of uses**

- Materials testing for thermal protection systems
- Gas pyrolysis (e.g. hazardous material breakdown)
- Material synthesis
- Coating deposition

**Plasmas in similar regimes are common**

- Torch may serve as a laboratory surrogate for plasmas observed in other applications
Science Questions

• ICP torch performance in applications depends on outlet plasma properties
  − What physical mechanisms control performance?
  − What level of physics modeling necessary for reliable predictions?

• Only operates stably over a restricted range of conditions
  − Predict limits of stable operation
  − What processes dictate these limits?
Simulation Milestone Objectives

- **Year 1**: Predict velocity & magnetic fields; Flow & EM-only scenarios
- **Year 2**: Predict exit profiles; Ar plasma, p = 1 atm, ṁ = 40 slpm
- **Year 3**: Predict exit profiles & stability; Ar plasma, Variable p and ṁ
- **Year 4**: Predict exit profiles & stability; Air plasma, Variable p and ṁ
- **Year 5**: Predict exit profiles & stability; Air plasma, Variable p and ṁ

- Add non-local Boltzmann
- Add air chemistry and RTE
- Model enhancements dictated by validation
ICP Torch Physics Overview

The torch represents a highly complex, multi-physics, multi-scale simulation challenge

Important Physical Phenomena

• Fluid & plasma flow
• Electromagnetism
• Non-equilibrium plasma dynamics
  – Ground- and excited-state neutral and charged species
  – Wide range of transformation/excitation reactions
  – Species and thermal transport
  – Heavies: Maxwellian translational energy distribution
  – Electrons: non-Maxwellian energy distribution
• Quasi-neutral plasma
• Participating media radiative energy transfer

Relevant Scales

• $D = 6\text{cm}, D_{\text{noz}} = 3\text{cm}, L = 25\text{cm}$
• Inlet jet width: 0.5mm
• $V \sim 10\text{m/s (chamber)}, 300\text{m/s (inlet)}$
• Turbulent scales near inlet: 10µ
• Excitation frequency: 6MHz
• EM wavelength: 50m
• EM skin depth: 6mm
• Wall sheath thickness: 50µ
• Electron mean free path: 7-70µ
• Heavies mean free path: 2µ
Resolution and Computation Requirements

**Straightforward Representation**
- Air feed gas, ~40 species (with excited states)
- $10^{11}$ grid points
- 4000 DOF/point for Boltzmann & RTE
- $10^{17}$ flop/step x $10^7$ steps
- 300 exaflop-hours

**For Industrial Applications**
- Perhaps 4 times bigger
- Complex hydrocarbon feed gas
- 1000 times computational cost

**Discretization & Algorithms Research**
- Multiple spatial grids
- New Boltzmann & RTE representations
- Multirate-timestepping
- $10^{11}$ DOF, 30 petaflop-hours

**Research challenges**
- Physical modeling
- Efficient discretizations
- Validation and uncertainty quantification
- Exascale computer science
- Exascale algorithms
Physical Modeling Approach

Initial Model Components

• Compressible Navier-Stokes (DNS)
• Darwin approximation to Maxwell's equations
• Multi-species transport, including excited states
• Mass action reaction kinetics
• Separate heavy species and electron energy equations
• Non-Maxwellian electron kinetics from local Boltzmann
• Optically thick radiative transport for resonant lines, otherwise optically thin

Refinements for More Challenging Cases

• 6-D Boltzmann for electron kinetics—needed to explore boundaries of stable operation
• Radiative transfer equation for lines that are neither optically thick nor thin—needed for air
• Other refinements as driven by validation
Uncertainty Modeling Approach

- Multiple sources of uncertainty
  - Device and scenario parameters (e.g., as built versus as designed geometry)
  - Model parameters (e.g., reaction rate constants, transport parameters, etc.)
- Expect uncertainty from modeling to dominate
- Relevant parameters can be decomposed
  - Characterization of interaction—collision and reaction cross sections
  - Representation of "environment"—distribution functions

\[ k_r = \int_0^\infty 4\pi g^2 \sigma^r(g) \chi_{kl}(g) dg \]

- Distribution function representation is target of planned model improvements and validation
- Within given model, uncertainty then traces to fundamental cross sections
- Calibrate stochastic cross section models using literature data
  - Collision and reaction cross sections (LXCat)
  - Oscillator strengths and photonic absorption cross sections (NIST)
Model Validation

• Most aspects of physical model may be validated outside of the torch via experiments accessing relevant plasma regimes
  – Electron kinetics
  – EM coupling of energy into the plasma
  – Wall sheath representation

• 6 MHz glow discharge device at $P \sim 1-100$ Torr gives electron density similar to torch, useful for validating
  – Electron kinetics
  – EM coupling of energy into the plasma
  – Wall sheath representation

• Mach 5 shock-induced non-equilibrium facility
  – Sub-mm spatially resolved measurements along stagnation line will be used to test models of non-equilibrium kinetics

• Full model will also be validated against torch measurements
  – Measure temperatures and densities in plasma chamber and jet exit
  – Determine limits of stable operations
  – Assess both models and validation process itself
UQ Algorithmic Research

Need new UQ methods and algorithms that overcome the challenges of (1) extreme computational cost of the forward simulation, (2) multiple disparate sources of uncertainty / high-dimensional uncertain parameters

- Multifidelity approaches
- Decomposition approaches
Multifidelity UQ

• Multifidelity methods: Leverage cheap approximate models to accelerate a UQ task

• Challenges:
  • number of evaluations of high-fidelity model is extremely limited
  • highly nonlinear, chaotic behavior
  • more complicated QoIs (e.g., locating the stability boundary)

• Approaches:
  • new approaches to exploit relationships among models, and with experimental data
  • leverage error estimates
  • leverage multifidelity sensitivity/adjoint information

We will exploit coarse grids, simplified physics, closure models, geometric reductions, data-fit surrogates, physics-based reduced models, ...
Decomposition Strategies for Scalable UQ
“Divide & Conquer”

- Challenges:
  - hundreds of parameters – need to identify “optimal” decompositions that may go beyond physical intuition
  - composition: re-integration of local component-level UQ to provably recover system-level UQ results

- Approaches:
  - build on Static Condensation Reduced Basis Method and Domain Decomposition UQ
  - exploit information from lower fidelity models (e.g., sensitivities) to identify decompositions
  - explore machine learning methods to identify decompositions
Exascale
Electron Boltzmann and Radiation

Electron Boltzmann equation

• 6D+time advection driven by collision integral
• Spatially adaptive discretization in both space (static) and velocity (dynamic), capitalizing on our work on fast spherical harmonics methods
• Treatment of advection: Both explicit as well as semi-Lagrangian for larger CFL, capitalizing on our work on scalable semi-Lagrangian methods on heterogeneous architectures

Radiative transfer equation

• 5D+quasi-steady in time with composition & temperature dependent coefficients
• Static space adaptivity with dynamic angle adaptivity, using coarser (than flow) space/time discretization, also exploiting fast spherical harmonics methods
• DG discretization and Krylov solvers for spatial operators
• Parallel sweeping methods (connection to graph algorithms)
Turbulent flow + species advection

- High order finite elements on unstructured meshes
- Locally hp-refined meshes for boundary layers and turbulent inlet
- Investigating multirate time stepping to handle wide range of flow scales (highly turbulent to laminar), requiring adaptations of our dynamic load balancing schemes
- Implicit solver for stiff component: employ our hybrid spectral-geometric-algebraic multigrid
  - may require specialized coarse grid solve via H-matrix approximation
  - adapt to CPU-GPU nodes
Core CS requirements for the Torch simulation

- Multiple physics, models, discretizations, and integration with V&V and UQ algos
- Algorithmic work optimality
- Performance and parallel scalability

- **Performance portability**
  - Faster algorithmic and model exploration
  - Minimize device-specific code
  - Improve maintainability
  - Fewer lines of code

- **Rapid development**
  - Interface with other frameworks
  - Yet, allow efficient integration

Existing HPC frameworks focused on performance portability challenge

But none provides what is needed for the Torch simulation
Our proposal: Parla

Goals: Rapid development, performance portability, interoperability

• MPI+’X’ (Parla)
  − MPI has solved the internode scalability problem
  − Per node performance and portability remains an outstanding challenge

• Python API
  − The most popular rapid development framework
  − Huge community / industry support
  − Extensive package support
  − 2018 Gordon Bell Prize, several GBP finalists (E.g., PyFR)

• Not monolithic
  − Leverages existing HPC efforts on performance portability
  − Enables interoperability/incremental adoption
Parla proposed components

**MPI + Parla**

**Parla Python API**
- Data and task parallelism / nd-arrays
- Memory placement/movement
- Parla Kernel Generator (PKG)

**Parla Runtime**
- Scheduler (task graph/data migration)
- Performance prediction / Synthesis
- Runtime interoperability

**Parla Productivity tools**
- Debugger
- Profiler/roofline analysis
- Code migration

PKG
Python → LLVM

Interoperability w/ ROCm, CUDA, Kokkos, RAJA, OpenMP, pthreads
Programming model

(a) Parla kernels

```python
@specialized
@jit(void(float64[:, :]), nopython=True, nogil=True)
def cholesky_inplace(a):
    for j in range(a.shape[0]):
        b = (a[:, j] * a[:, j]).sum()
        a[j, j] = sqrt(a[j, j] - b)

    for i in prange(j+1, a.shape[0]):
        a[i, j] -= (a[i, j] * a[j, j]).sum()
        a[i, j] /= a[j, j]
```

```python
@cholesky_inplace.variant(gpu)
def cholesky_inplace(a):
    a[:] = cupy.linalg.cholesky(a)
```

```python
@cholesky_inplace.variant(fpga)
def cholesky_inplace(a):
    ca = fpga(a)
    ca = fpga_cholesky(ca)
    a[:] = ca
```

(b) Parla tasks

```python
def cholesky_blocked_inplace(a):
    T1 = TaskSpace("T1")
    T2 = TaskSpace("T2")
    T3 = TaskSpace("T3")
    T4 = TaskSpace("T4")

    for j in range(a.shape[0]):
        for k in range(j):
            @spawn(T1[j, k], [T4[j, k]])
            def t1():
                a[j, j] -= a[j, k] @ a[j, k].T
            @spawn(T2[j], [T1[j, 0:j]])
            def t2():
                cholesky_inplace(a[j, j])
                for i in range(j+1, a.shape[0]):
                    for k in range(j):
                        @spawn(T3[i, j, k], [T4[j, k], T4[i, k]])
                        def t3():
                            a[i, j] -= a[i, k] @ a[j, k].T
                        @spawn(T4[i, j], [T3[i, j, 0:j], T2[j]])
                        def t4():
                            ltriang_solve(a[j, j], a[i, j].T)
                return T2[j]
```
C++

w/ CUDA
Parla usage plan

- C++ main
  - MFEM or Albany
  - Time stepper (operator split)
  - Multiphysics

- Fluids - MFEM or Albany
- Boltzmann
  - Parla
  - mpi4py
- Maxwell - MFEM or Albany

Performance-Portable Kernels
- semi-Lagrangian advection
- FFT/BLAS
- Collision / Sweeps
- Custom sparse direct solvers

- Python: Numba / PKG
- C++: Kokkos or RAJA +
  - Cython or PyBind11

Vendor libraries
- Kokkos kernels
- ASICs/FPGAs
Current state of Parla

• Working prototype library that supports multicore and multi-GPU nodes
• Python implementation with online task scheduling
• Has been exercised at TACC
• Docker support
• Working examples for
  - Fox’s matrix-matrix multiplication
  - Inner product
  - Recursive Block Cholesky
  - Simple Jacobi iteration
  - MPI interoperability

https://github.com/ut-parla/Parla.py

Developers
Ian Henriksen, Oden & CS
Arthur Peters, CS
Will Ruys, Oden
Software Development

- **Overarching Software Goal:** Develop performance-portable software for complex multi-physics modeling

- We will rely on rigorous development pipeline using modern software engineering practices to achieve full-system predictions in Year 2

- High-level application development:
  - C++ + Parla + MPI + Support libraries/runtimes

![Diagram showing modeling schema interactions]
Development Environment

• An important evolving technology allowing for reproducibility, portability, and replication of underlying software environments is application containerization
  - allows us to mimic supercomputing OS envs and stacks for local development
  - improves application portability

• Plan to maintain project-specific Docker images that include all underlying software libs and tools required for ICP torch application

• Also utilizing HPC-centric containers for execution on production resources
  - Singularity
  - Charliecloud

• Companion effort to periodically benchmark containerized app version against bare-metal
Development Environment (cont.)

• Project Management: GitHub
  - milestone/issue tracking
  - release management
  - version control (git)
  - wiki

• Continuous Integration (CI):
  - Travis (community instance)
    • compilation
    • unit testing
    • smaller regression tests
  - Jenkins (local instance, 1K-2K cores)
    • OpenHPC based
    • vendor compilers and math libraries
    • include larger regression tests

• Code Coverage Tools:
  - gcc/lcov + coverage.py

• Source Code Documentation:
  - Doxygen (C++)
  - Sphinx (Python)
Additional Supporting Software

- **Albany** and **MFEM** – FE discretization and numerics
- **BOLSIG+** and **Antioch** – chemical kinetics and transport
- **Trilinos** and **PETSc** – Krylov solvers, AMG
- **MASA** - manufactured solution verification library
- **Dakota** – optimization, sensitivity, and UQ analysis
- **QUESO** and **hIPPYlib** – Bayesian UQ support
- **libGRVY** – HPC utility library
- **PAPI**, **TAU** - detailed performance profiling
- **HDF5** – I/O library

**DoE driven efforts**

**Local UT efforts**
Summary

• Plasma torch physics
• Exascale numerical algorithms
• Core CS: Parla, compiler, and runtime, software productivity tools
• Cutting-edge software engineering

• The real torch!
## Senior Investigators

- **Robert Moser** (PI): Modeling, validation and uncertainty quantification
- **George Biros** (Co-PI): Exascale algorithms and software

<table>
<thead>
<tr>
<th>Name</th>
<th>Role/Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Noel Clemens</td>
<td>Experiments</td>
</tr>
<tr>
<td>Mattan Erez</td>
<td>Exascale runtime systems</td>
</tr>
<tr>
<td>Omar Ghattas</td>
<td>Exascale and UQ algorithms</td>
</tr>
<tr>
<td>Milos Gligoric</td>
<td>Exascale tools</td>
</tr>
<tr>
<td>David Hatch</td>
<td>Plasma physics</td>
</tr>
<tr>
<td>John McCalpin</td>
<td>Performance optimization</td>
</tr>
<tr>
<td>Tommy Minyard</td>
<td>HPC testbeds</td>
</tr>
<tr>
<td>Tinsley Oden</td>
<td>Uncertainty quantification</td>
</tr>
<tr>
<td>Todd Oliver</td>
<td>Modeling, validation, uncertainty quantification</td>
</tr>
<tr>
<td>Keshav Pingali</td>
<td>Exascale tools and the Parla (parallel) language</td>
</tr>
<tr>
<td>Laxminarayan Raja</td>
<td>Plasma physics and modeling</td>
</tr>
<tr>
<td>Christopher Rossbach</td>
<td>Exascale runtime systems</td>
</tr>
<tr>
<td>Karl W. Schulz</td>
<td>Modeling, exascale, performance analysis, software engineering</td>
</tr>
<tr>
<td>Philip Varghese</td>
<td>Kinetics modeling</td>
</tr>
<tr>
<td>Karen Willcox</td>
<td>Uncertainty quantification and UQ algorithms</td>
</tr>
<tr>
<td>Martin Burtscher</td>
<td>(Texas State University): Heterogenous architectures</td>
</tr>
</tbody>
</table>