Software Productivity Application: Integrated Modeling for Fusion Energy

David E. Bernholdt, Wael R. Elwasif, Samantha S. Foley*, and Donald B. Batchelor
Oak Ridge National Laboratory
Contact: bernholdtde@ornl.gov

* Current affiliation: U. Wisconsin-La Crosse

This work was performed in part at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC for the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.
Fusion Integrated Modeling, ca. 2005

- Plasma physics has a wealth of “single physics” codes
  - Multiple methods for the same physics
  - Different physical fidelity and predictive skill
    - “Reduced models” to detailed physics
  - Varying cost and scalability
  - Closely connected to experimental programs

- Coupled simulation experience limited
  - Primarily “sneaker-based” coupling
  - Primarily strong separation of time scales

Major U.S. Toroidal Physics Design and Analysis codes
Productivity for Integrated Modeling?

• Design for generality – nothing specific to fusion
• Focus on loosely-coupled time-stepped multiphysics simulations
• Operate in HPC environments, including leadership-class
• Support rapid exploration of new ideas – improve “time to science”
• Use *existing* “single-physics” codes as-is
  – Insufficient funding to support rewrites
• Use existing “single-physics” codes *as-is*
  – Codes under continual development, don’t want to fork
  – Most code owners will accept small changes
• Support multiple interchangeable implementations of the same physics
• Support components from outside the project
  – Make it easy to turn physics codes into components
• Make it easy to program “drivers”
  – Language and programming model should be comfortable for physicists
Choices Have Consequences

- Wide variation in performance and scalability of existing codes
  - Internally manage resource pool to provide flexibility to components without having to go back to system scheduler
  - Provide flexibility in how concurrency can be expressed and utilized within and across components

- Existing codes use files as their inputs and outputs
  - Make it easy to manage sets of files associated with components
  - Provide a single “central” store for data exchanged between components

- Available “workflow” tools target distributed rather than HPC environments
  - Develop a new framework with the needed HPC features
IPS Framework Basics

- IPS = Integrated Plasma Simulator
- Python-based component framework
- Components are python-wrapped binaries
- Framework runs in a batch allocation, manages resources for components
- Components launched using standard system mechanisms, i.e. mpiexec, aprun
- “Plasma State” holds primary data for exchange
  - “Reader-makes-right” model
Plasma State

- Overloaded term
- IPS has a generic “state” concept
  - Data (files) used by multiple components
- PPPL Plasma State File
  - netCDF-based file format
  - Well-defined (but easily extended) set of variables
  - Provides functions for grid interpolation, etc.
  - Store as produced, “reader makes right”
  - Used directly by some components, through helpers by others
IPS Component Architecture

Components
• Components characterized by ports (class) and implementation (instance)
  – All implementations of a port are expected to be fundamentally equivalent in their interactions with other components
• Primary component interface
  – init()
  – step()
  – finalize()

Driver Component
• Python script
  – Easy for physicists
• “Standard” driver covers most uses

Framework Services
• Configuration manager
• Task manager
  – Launch underlying applications
  – Blocking or non-blocking
• Resource manager
  – Nodes allocated to batch job
• Event service
  – Asynchronous pub/sub events
• Data manager
  – File staging (per timestep)
  – Mediate concurrent access to plasma state
  – Checkpoint/restart (framework level)
• Monitoring
  – Progress tracking via web portal
Multi-Level Parallelism in the IPS

1. Individual “tasks” (physics executables) can be parallel
2. Components can launch multiple tasks
3. Multiple components can run concurrently
4. Multiple independent simulations can run concurrently
A Design Target for Multi-Level Parallelism

Notable features

- Two highly scalable components (AORSA, NUBEAM)
- Many limited scaling components
- Run transport, Fokker-Planck, and stability analysis tasks concurrently
- Launch multiple analysis tasks (i.e. for each flux surface)
Improving Resource Utilization through Concurrency

Simple ITER modeling scenario
TORIC+NUBEAM+TSC

16 p, ~97 s
4 p, ~115 s
1 p, ~130 s

Number of concurrent simulations

Run multiple simulations in the same allocation

Run components concurrently within a single simulation

<table>
<thead>
<tr>
<th>Levels of Parallelism</th>
<th>Components</th>
<th>No. of Cores</th>
<th>Time (hrs)</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task only</td>
<td>TORIC + NUBEAM + TSC</td>
<td>16</td>
<td>28</td>
<td>448</td>
</tr>
<tr>
<td>Task + Component</td>
<td>TORIC + NUBEAM + TSC</td>
<td>24</td>
<td>12</td>
<td>288</td>
</tr>
<tr>
<td><strong>Change</strong></td>
<td></td>
<td><strong>+150%</strong></td>
<td><strong>-43%</strong></td>
<td><strong>-64%</strong></td>
</tr>
</tbody>
</table>

Levels: Task only, Task + Component
IPS as Execution Engine for DAKOTA

- DAKOTA (SNL)
  - Parameter sweeps
  - Design optimizations
  - UQ

- IPS provides better support for resource management and parallelism

- Use IPS to execute simulations under control of DAKOTA

![Graph showing processor utilization for 9-simulation parameter scan]
Event-Based Execution Models Also Supported

- Event service provides asynchronous publish/subscribe messaging between components
- Pull model to avoid requiring threading within components
- Payloads are up to components
  - Control signals
  - (Modest) data payloads
Parareal Algorithm (Parallel-in-Time)

- Predictor-corrector, iterative method for time-dependent PDE's
  - Lions, Maday, and Turnici, 2001
- Advance system state from initial condition $\lambda_0$ at time $t_0$ to time $t_f$, using $N$ time “slices” (sub-intervals), each of size $\Delta t$, where $T = t_f - t_0 = N \Delta t$
- **Fine**, accurate (expensive) solver, $F$ compute “true” solution
- **Coarse**, approximate (fast) solver, $G$, compute approximate solution
- **Convergence** tester, $C$
- Guaranteed convergence in $K \leq N$ iterations
- “Good” scenarios have $K << N$

\[
F \equiv \frac{du}{dt} - \lambda u = \sin(10\pi t)
\]
\[
G \equiv \frac{du}{dt} - \lambda u = 0
\]

$k = 3, N = 10$
Task-Based View of Parareal

Tasks can execute as soon as dependencies are complete

See talk Coarse Grained Task-Based Parareal Parallel-In-Time Applications in Fusion Energy, Wael Elwasif and Debasmita Samaddar, @ 2:20pm in room 254 B
Part of MS171: Task-based Scientific Computing Applications - Part II of II, 1:30-3:10pm
Implicit Coupling of Discrete Components

- Minimal Polynomial Expansion
- Conceptually: centered Crank-Nicholson that can be implemented with quantities available from existing codes
- Like a Krylov subspace method for non-linear systems

\[ E_{i+1} \rightarrow AORSA \rightarrow CQL3D \]
\[ D_{i+1} QL \]
\[ f_{i+1} \]
\[ 1/2(D^i + D^{i+1}) \]

Least Squares

Iterate \( j = 0 \ldots 5 \)

Work by F. Jaeger, E. D’Azevedo, J. Wright, A. Bader

Diagram:
- CQL3D
- AORSA
- Least Squares
- MPE Implicit
  - 5 steps 2.3 hrs
- Explicit
  - 125 steps 8hrs

Graph:
- Hydrogen energy at rho = 0.1

2015-03-15 SIAM CSE 15

Hydrogen energy at rho = 0.1

MPE Implicit
- 5 steps
- 2.3 hrs

Explicit
- 125 steps
- 8hrs
IPS as a Platform for Resilience Research

Work supported by Coordinated Infrastructure for Fault Tolerance Systems (CIFTS) project (DOE ASCR)

- Node failures
  - RM marks node out of service and re-executes task on other resources
- Out of memory error
  - TM reconfigures request to provide more memory per process, re-executes
- Segmentation fault
  - Abort simulation, let developer fix code
  - Re-execute to tolerate known bug

Work with Aniruddha Shet and Samantha Foley
Monitoring Job Progress

- Real-time job monitoring from anywhere
- Visualization data in browser
- http://swim.gat.com:5050
IPS Uses Outside of Fusion

- **Virtual Integrated Battery Environment (VIBE)**
  - DOE-EERE CAEBAT Program
  - Integrated Li-ion battery modelling
  - Electrochemical – thermal – electrical – mechanical phenomena
  - Cell sandwich → cell → pack → module

- **Multi-hole Injector Optimization for Spark-Ignited Direct-Injection Gasoline Engines**
  - w/ General Motors Research
  - DAKOTA + IPS for parametric scans and optimization
  - Nozzle flow CFD + external spray simulations

Streamlines of velocity magnitude of flow through an injector nozzle showing the onset of vapor formation (pink iso-surfaces)

Temperature in 4P and 4S module with fully coupled electrochemical, electrical and thermal simulations in CAEBAT OAS/VIBE

Experimental Results

Computational Model
Summary

• The IPS is a simple Python component framework designed to facilitate the execution of loosely-coupled multiphysics simulations and similar workflows
  – Driven primarily by fusion requirements, but designed for generality

• Provides an effective way to work with existing physics codes, as-is
  – Accept certain limitations

• Internal resource management + four different levels of parallelism + events
  – Accommodates disparate scalability and performance characteristics of different components

• Provides a platform for rapid exploration/new research
  – New simulations, algorithms, implementations, computer science issues

• Helped plasma physicists carry out simulations they would not have previously considered tractable

• “Productivity” is often driven by very local context (project, domain, timeline) – where are your pain points today?