

Update on IPS design and workflow

D. B. Batchelor – ORNL

Sept. 25-27, 2006

Oak Ridge National Laboratory

stuff

Elements of the integrated Plasma Simulator

Simulation directory

/IPS_run_XYZ/

/Simulation_setup/
Simulation definition
Initial plasma state
Code specific initialization

/work/
Current plasma state data
<Component>_/work/
Configuration files
Scripts and executables
Input files
Code scratch files
Output files
<Other components...>

/Simulation_results/
Plasma state history
Current component restart
Component history

Simulation Driver

IPS Driver Script

Computer environment initialization

Plasma state initialization

Initialization of all components

Advance Sequence: [t → t+Δt]

Evaluate Simulation Control Logic

Step all components (e.g. RF_step())

Evaluate results of time step
(terminate, or adjust and restart current step, or accept current time step)

Commit data from time step to /Simulation_results directory

Finalize simulation

Typical component

<Component> script

<Component>_initialize

<Component>_step
prepare <code> input
run <code>
process <code> output

<Component>_finalize

Framework

CS services
Portal
Job manager
Data manager
Event channel

Physics services
Re-gridding
Time interpolation
Mathematical libraries
...

Plasma State Component Functions

PS_GET_PLASMA_STATE
PS_STORE_PLASMA_STATE
PS_COMMIT_PLASMA_STATE

IPS Driver Script

Computer environment initialization – set up directories, do computer science stuff

Plasma state initialization – set up the “current” plasma state for t_0

Initialization of components – do loop, call `<COMP>_INITIALIZE` functions

Loop over time:

Take a time step, $t \rightarrow t + \Delta t$

Evaluate pre-step control logic – general logic relevant to all components

Cycle through all components:

Evaluate pre-step control logic relevant to this component – component specific logic (e.g. changes in heating power)

Step component – call `<COMP>_STEP` function

Evaluate post-step control logic relevant to this component – evaluate what to do about any errors or warnings returned from the `<COMP>_STEP` function

End cycle through components

Evaluate Post-step control logic – general logic after all components have been stepped

Evaluate validity of time step – step ok?, need to make adjustments and re-run step?, give up and terminate?

Accept time step – CS move files to `/results/` directory (maybe just links)

End time step

End loop over time

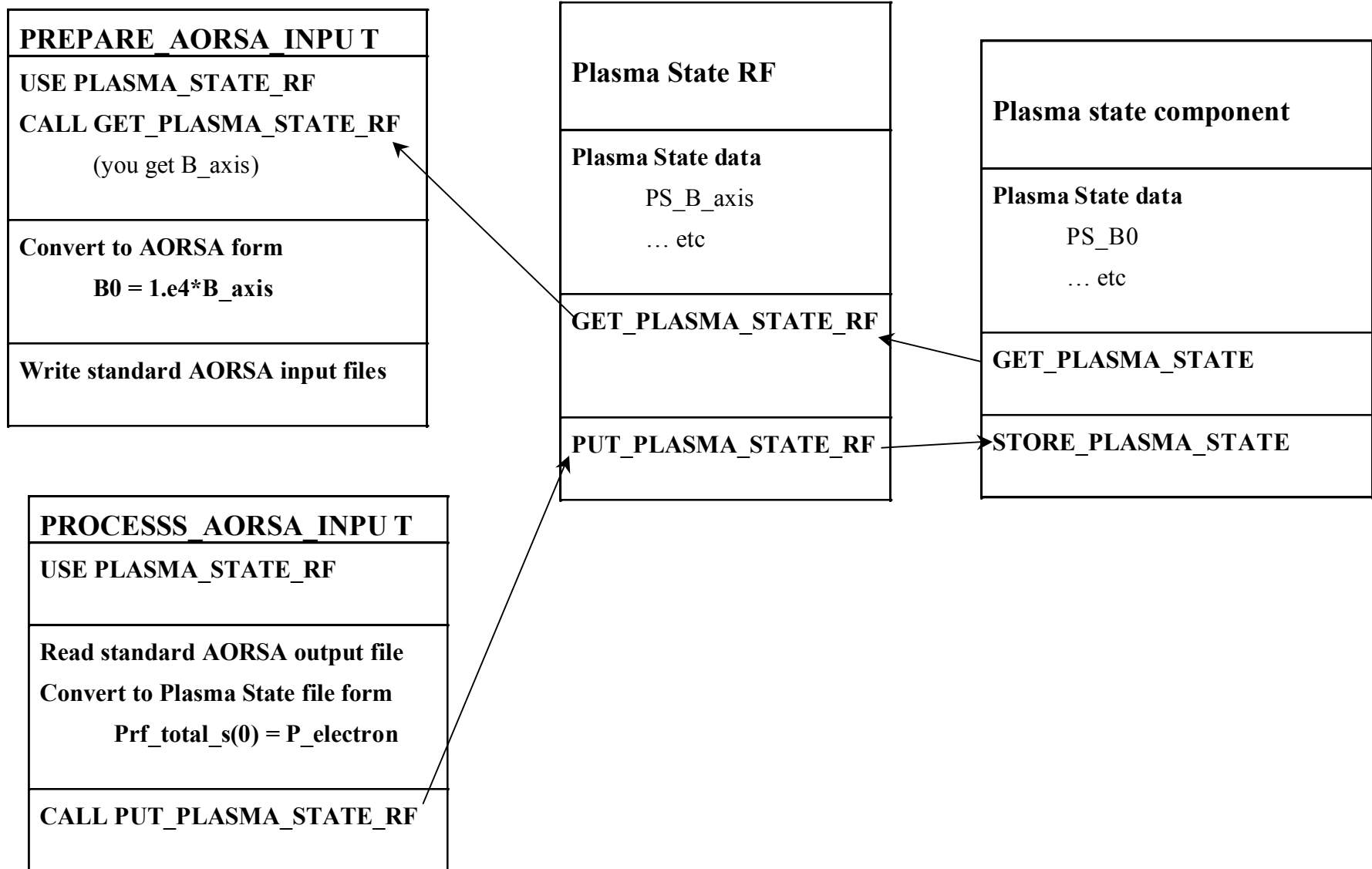
Finalize simulation – put files in order, call `<COMP>_finalize` functions

Each component exports three functions

Component Script (e.g. <COMP>_A)
Initialize function – e.g. A_INITIALIZE(err_info)
Time step function – e.g. A_STEP(err_info)
Finalize function – e.g. A_FINALIZE(err_info)

RF SOLVE_STEP
<p>CS work</p> <p>Make connections with framework services (data manager, job manager, portal, etc)</p> <p>Make new run directory for this time step</p> <p>Other magical CS stuff</p>
<p>Prepare code input → Call PREPARE_AORSA_INPUT (fortran executable)</p> <p>Call GET_PLASMA_STATE_RF – loads relevant plasma state data into module variables</p> <p>Read AORSA specific data from input files</p> <p>Combine PS data and other AORSA input data and write standard AORSA input file</p> <p>CS notify file events</p>
<p>Run code → call RUN_AORSA</p> <p>Check to be sure everything needed for AORSA run is ready</p> <p>Launch AORSA executable</p> <p>CS notify run event, file events</p>
<p>Process code output → call PROCESS_AORSA_OUTPUT</p> <p>Evaluate results of AORSA run – if run ok proceed, if not then:</p> <p>Evaluate control logic – Are there adjustments that can be made to AORSA inputs that might make the run succeed. If so then make the changes and re-launch AORSA</p> <p>If nothing can be done inside the component to fix the step, report this situation back to the driver script (error return) and quit</p> <p>Update Plasma State</p> <p>Call PUT_PLASMA_STATE_RF – stores RF results into Plasma State</p> <p>Write AORSA internal state in AORSA_restart_file (for AORSA there actually is no internal state)</p> <p>CS notify step success or failure, file events, clean up file mess</p>

Communication with Plasma State



Proposed new Plasma State interface

- **For every data structure supported in the plasma state there will be accessor get and put fortran90 subroutines e.g.**
 - PS_put_real_scalar(“B_axis”, B_axis)
 - PS_get_real_scalar(“B_axis”, B_axis)
- **For every data structure which discretizes a continuous function there will be additional arguments describing the grid and the recommended method to construct the continuous representation (e.g. interpolation)**

For example the species particle densities

- PS_put_profile(“n_s”, n_s, nrho_n, rho_n_grid, “conserving_bi_cubic_spline”)
- PS_get_profile(“n_s”, n_s, nrho_n, rho_n_grid, interp_method)
- **These shall be bundled into two get and put generic subroutine interfaces e.g.**
 - PS_put_(“B_axis”, B_axis)
 - PS_put_(n_s, nrho_n, rho_n_grid, “conserving_bi_cubic_spline”)
- **Facilities shall be provided as a framework service to interpolate from the native grid in the Plasma State to other grids. These must support all of the representation methods recommended in in the put calls of the plasma state. e.g.**

```
subroutine interpolate(old_profile, old_grid, new_profile, new_grid,  
interp_method)
```

Proposed new Plasma State interface (2)

- **For every data structure supported in the plasma state there will be accessor get and put fortran90 subroutines e.g.**
 - `PS_put_real_scalar("B_axis", B_axis)`
 - `PS_get_real_scalar("B_axis", B_axis)`
- **For every data structure which discretizes a continuous function there will be additional arguments describing the grid and the recommended method to construct the continuous representation (e.g. interpolation)**

For example the species particle densities

- `PS_put_profile("n_s", n_s, nrho_n, rho_n_grid, "conserving_bi_cubic_spline")`
- `PS_get_profile("n_s", n_s, nrho_n, rho_n_grid, interp_method)`
- **For every data structure which discretizes a continuous function there will be additional get functions that encapsulate interpolation to construct the continuous representation**
 - `PS_get_profile_interpolated("n_s", n_s, nrho_n, rho_n_grid, interp_method)`
 - **In this case the arguments `nrho_n`, and `rho_n_grid` are intent in arguments describing the desired interpolated grid**
- **These shall be bundled into generic subroutine interfaces e.g.**
 - `PS_put_("B_axis", B_axis)`
 - `PS_put_(n_s, nrho_n, rho_n_grid, "conserving_bi_cubic_spline")`

Other points

- **SWIM_global_data_mod.f90**
- **Kruger's fluxgrid as a framework service?**

IPS design – Component based architecture, Plasma State component plays a central role, components implemented using existing fusion codes

