

Use this guide to run the ARIES systems code (ASC)

Lane Carlson

v.1 2010-01-05

v.2 2010-04-15

v.3 2011-03-07

v.4 2012-01-26, ASC SVN v.34 new input files

Please see the ASC Homepage for more details: <http://aries.ucsd.edu/carlson/aries/>

Files that compose the ASC:

- The ASC is composed of a physics module, an engineering module, and various input and output files. This guide assumes that you have all the ASC files necessary to run the code. A more in-depth explanation of the files can be found on the Documentation page of the ASC Homepage.
- **Physics module:** *inpar.txt* & *inpar2.txt* (input files), *corona* (plasma equilibrium look-up table), *makefile* (instructions to build the executable), *PlasmaPhysics.exe* (executable), *runphysics.sh* (run command shell), *sysplas_PC.F* (source code).
- **Engineering module:**
 - **Input data:** *builds_generic_sic.data* or *builds_generic_dcil.data*, *costing.data*, *inflationfactors.data*, *magnets.data*, *materials.data*, *powerflow.data*, *setup.data*, *sysoutsic.data* or *sysoutdcil.data*
 - **Core Code:** *Aries.cpp*, *AriesSystemsCode.exe*, *Cost.cpp*, *Cost.h*, *CostingAccount.cpp*, *CostingAccount.h*, *DesignPoint.cpp*, *DesignPoint.h*, *Geometry.cpp*, *Geometry.h*, *Part.cpp*, *Part.h*, *runaries.sh* (SCLL), *runaries2.sh* (DCLL).

1. Run the PlasmaPhysics module on a local computer or remote cluster computer:

- Log in to PPPL cluster via FTP (to transfer files, portal.pppl.gov) and Terminal (to issue commands, `ssh -l lcarlson portal.pppl.gov`).
- Modify the **inpar2** file to your liking. **Inpar** contains constants if the scanning range is set to 1.
- **Makeclean**, then **make** to construct the executable, clear the log file (on the cluster)
- Then run in terminal `./runphysics.sh > log.txt &` (& allows it to run in the background)
- To run on cluster, `qsub plasmaphysicsscript.sh` (use `stat -u lcarlson` to view status)
- This outputs the data to **sysout**. Each line has 55 variables describing the plasma parameters.
- Computation Time: see below.

2. Generate system models:

- Add **.data** to end of **sysout** and remove the first line of the file, which indicates the number of points, then save. Log this information in the ASC *Legend #.xls* chronicle spreadsheet, including inpar files used with scanning ranges.
- Copy **sysout** to AriesSystemsCode/SiC/InputData folder.
- Clear the log file in main folder and SysoutFinal.data in COEMaps.
- Choose between **coe** (production mode for finding COE for *many* points) and **system** (detailed geometry and analysis for a **single** point) in *setup.data* file.
- The setup file also allows different selections of the divertor and blanket to use.
- Other setup files give input data for costing, materials, and radial builds.

(The difference between running the code in "**system**" and "**coe**" mode is that in the former case the code prints the geometry, costing accounts and lots of other data for a single data point for the purpose of debugging, etc. whose output is in OutputParts subdirectory. In case of "coe" mode, it is assumed that you want to plot many data points together, so all the unnecessary geometrical plots and other details are skipped and a more concise output is generated in COEMaps/SysoutFinal.data.)

- Specify data range in **runaries.sh** file for number of points to look at. Use Matlab **"scriptbreaker.m"** program to divide up the work evenly between the nodes. *Remember to subtract one from the line number since the first line is counted as zero.*
- Need to **make clean** to clear the executables, then **make** if any of the .cpp files are modified.
- Then run in terminal **./runaries.sh > log.txt &**
- The output files for a "system" run are put in /OutputParts.
- The raw output file is in **COEMaps/SysoutFinal.data**, whose location is specified at the bottom of Aries.cpp. Each line has 86 variables that describe the operating point.

Computation Time:

- MacBook laptop, 2GHz, Core 2 Duo, 2 GB ram = ~700 /hr
- PPPL Cluster, AMD quad-core node, 1 CPU, 2 GB ram = ~31k /hr/core (physics), ~2k /hr/core (engineering)
- See **"benchmarking cluster note.rtf"** for more cluster benchmarking tests.

3. Run the engineering module on PPPL cluster (<http://beowulf.pppl.gov/>)

- First need to upload the module to the PPPL cluster, at vpn.pppl.gov in the **/u/lcarlson/sic** folder. If the VPN does not work, use **auth.pppl.gov**.
- Need to break up the runaries.sh range to compute by considering how many cluster computers you want to employ. Look at how many points sysout.data has. The computation time depends on the number of nodes you employ. Each node of the cluster can process ~1900-2000 pts/hr (ASC v.30), ~600-800 pts/hr (ASC v.34).
- Use Matlab **scriptbreaker.m** to break up and distribute the points evenly between the nodes. This reads the database and computes how many points each node should process.
- You will need as many **runaries.sh** and **ascscript.sh** scripts as you have nodes. You **MUST** submit a **queue**, not just execute the code from the portal.
- **NEW scripts** - use Matlab m-file **runaries_printer.m** to input the node properties, number of files, etc you want and it will print the **runaries.sh** and **vasstscript.sh** and **submitall.sh** scripts. The long way is to modify each text file by hand - use a template depending on the number of points you have, then break up the task into 10-20 jobs for faster computation time.
- In the script shell **"runaries#.sh"**, use **1 node, 1 CPU, 1 GB RAM, ~72 hr processing time**, and specify the **kruskal or ellis** nodes - they seem to start running right away. More info available on node benchmarking in a **benchmarking cluster note.rtf**. The code is not optimized to take advantage of parallel processing; the only advantage is to use multiple cores/computers in parallel.
- Put the **vasstscripts#.sh** queuing scripts in the main engineering folder, as well as the **runaries#.sh** scripts and the single **submitall.sh**.
- See **2.** above for the details on how to set up all the running of the code.
- Note that the output files will all dump randomly as they finish into one file in COEMaps. This is OK since the points will be filtered later. They needn't be in order. It is also ok if there are redundant points because the post-filtering will take care of them later.
- Delete all unnecessary files and .svn files to give the account maximum space. Limit is 10 GB on the home directory **/u/lcarlson**, but also have a 100 GB project disk at **/p/vasst**.
- Ensure that **InputData** contains **costing, magnets, powerflow, and sysout.data (plus more)**.
- Ensure that the ASC does not print out PartCompositions, etc. Need to comment these sections out in the code.
- Remember to **compile ("make")** the code on the PPPL cluster, via the VPN terminal. The code will not run on the cluster if it was compiled on your local machine.
- Change the **file permissions** for the folder after you upload files or else you can't compile. They are automatically set to read-only and need to be changed to read-write by the command **chmod ugo+rw * Use ls -l** to check file permissions are **rw**.
- To submit the scripts for computation on the cluster, you must submit each one with **"qsub vasstscript#.sh" (going from 01 to # nodes)** or use **"submitall.sh"**

- Now sit back and let the nodes crank out the data. You can set the *vasstscripts* to have the cluster send you an email when the job is done.
- Helpful Unix commands:

<i>du -h</i>	disk usage
<i>qsub <runscript.sh></i>	submits job to cluster queue
<i>qstat -u <username></i>	shows jobs running under user
<i>fnodes</i>	number of free nodes on the cluster
<i>showbf</i>	check nodes in use
<i>chmod ugo+rw *</i>	changes all files to read/write access
<i>qdel <job ID></i>	deletes job from queue
<i>kill <job ID></i>	stops the process
- When **finished**, copy the SysoutFinal data to **/p/vasst** project disk for backup. **cp** or **mv** **name.data /p/vasst/sic/**

4. Filtering and Plotting

- For filtering and visualizing the data, see the next document "**How to run the VASST GUI v.#**"

5. Finding a specific point in the database, necessary for selecting the strawman point

- Once a desirable point is found to select as a strawman, then that point has to be selected from the compilation of raw databases. For this, use **findline.m** to find the point you want in the filtered database from the raw db.