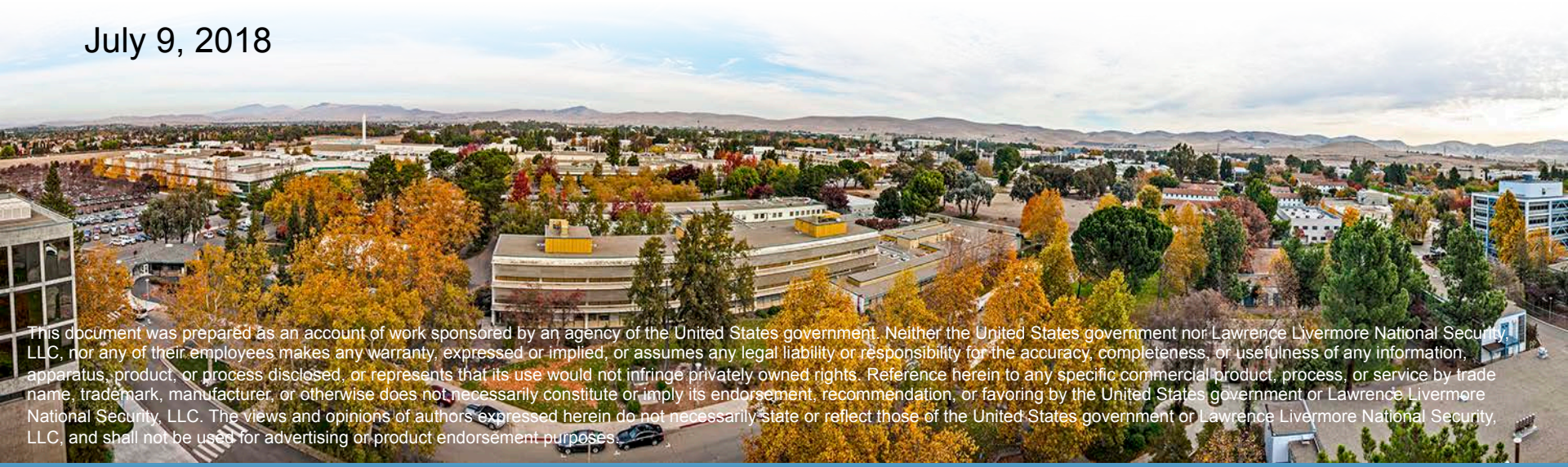
CRETIN

Session 10

Howard Scott

July 9, 2018



LLNL-PRES-755118
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

Lawrence Livermore National Laboratory

Session topics

Postprocessing rad-hydro simulations

- Overview
- Postprocessing workflow
  1. Extracting data from the rad-hydro code
  2. Inserting data into Cretin
  3. Populations
  4. Spectral opacities / emissivities
  5. Detector spectra

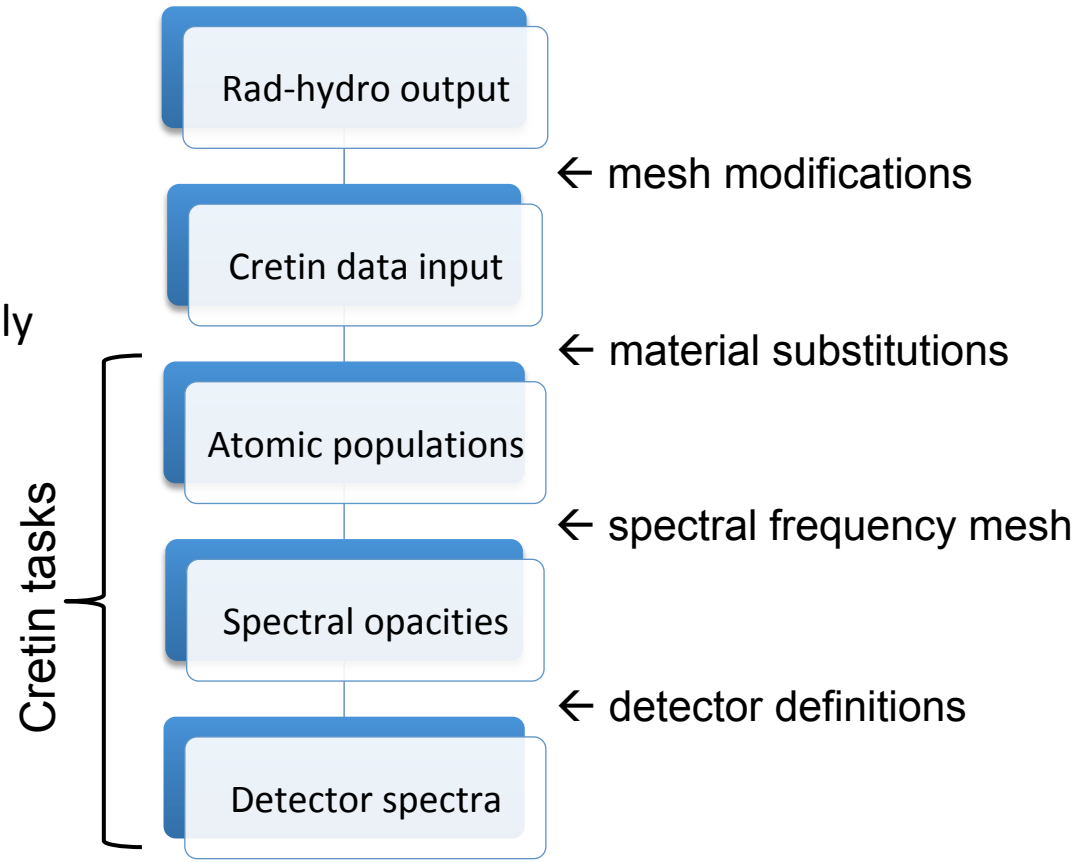
Overview

- Rad-hydro codes can do the "same" physics as Cretin
  → NLTE atomic kinetics + radiation transport
- Why postprocess rad-hydro simulations with Cretin?
  1. Increase spectral resolution to simulate diagnostics
  2. Decrease spatial / temporal resolution to restrict expensive NLTE physics to a limited number of zones and/or timesteps
  3. Substitute atomic models or different materials
  4. Improve detailed physics, e.g. line transport, line shapes
- Approximations / gotchas
  1. Time evolution in Cretin assumes Lagrangian material
     → most postprocessing uses steady-state kinetics
  2. Cretin uses a logical node-centered mesh, unlike most rad-hydro codes

Main assumption: "improved" physics would not significantly change the rad-hydro results

Workflow considerations

- Multiple distinct tasks
- Each task adds additional information (with a chance to make modifications)
- Tasks can be done independently (in serial)
- Timesteps can be done independently if steady-state treatment is valid
  → parallel simulations



Separating the Cretin tasks can be convenient, but is not necessary

Extracting data from rad-hydro codes

Required data:
mesh position and velocity: $\mathbf{r}, \mathbf{v}$
material temperatures and densities: T_e, T_i, ρ
element number densities: n_{iz}

Optional data:
electron density, ionization: n_e , $<Z>$
photon spectrum: J_ν
absorption / emission: κ_ν, η_ν

- Restart dumps will have <u>all</u> required data
- Opacity / visualization dumps must request n_{iz}
  - <u>not</u> included by default
  - alternative for a Lagrangian run is ρ + element concentrations
- Yorick files from ARES must request all desired data

Pay attention to the time spacing of dumps used for postprocessing

Inserting data into Cretin

- Rad-hydro data comes in through xfiles
- Available Yorick scripts read dumps, process data, and produce an xfile
  - Scripts are available for HYDRA, ARES, CALE, (LASNEX), … in /usr/apps/cretin/yorick

    hydra2cr.i, ares2cr.i, cale2cr.i … + xfile-util.i
  - Use restart (or other) dumps from HYDRA, CALE, LASNEX or Yorick dumps from ARES
- All requested timesteps / cycles go into a single xfile
- Most variable names are code specific, some can be specified – use "help" for details
- User may need to provide axis identification
- Example:

```
yorick -i hydra2cr.i
xf_axis = [1,3]
xf_tmin = 2.e-9
> hydra2cr, "hydr", "hyfile"
Constructing xfile …/hyfile.xfa
processing file hydr11256 time = 2.022e-09
        . . .
```

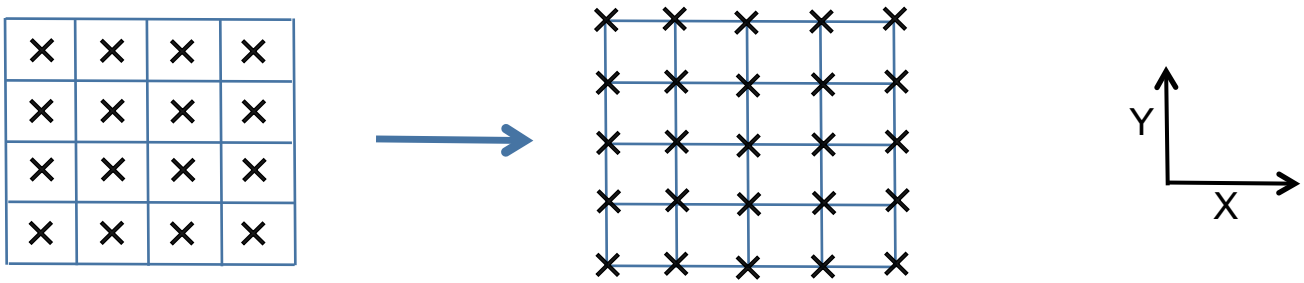
Identify HYDRA (x,z) axes as Cretin (x,y) or (r,z)

Set start time (in sec) for processing dumps

Process dumps starting with "hydr"

Notes on processing data

- Final mesh for Cretin must be a node-centered logical mesh
- Yorick scripts change data from zone-centered to node-centered



- Multiblock meshes will produce an xfile for a specified block – or one per block (but using multiple blocks in a Cretin run is not easy)
- Other options:

  reflecting / extending / restricting mesh + data
  identifying / restricting elements included in xfile
  breaking up element number densities by region (see *xf_rnuc*)

Yorick scripts use *xfile-util.i* as a backend – both files are required to maintain consistency

Considerations for producing a Cretin mesh

- Spatial resolution used by the hydro code may be excessive for Cretin
  - computational time / memory requirements for atomic kinetics / radiation transport may be prohibitive
  - kinetics parallelizes well across compute nodes, but rad transport does not
- Distorted mesh near axis in rz-geometry can cause difficulties

  → if re-meshing to decrease resolution, use an orthogonal mesh (near axis)

- Resolution requirements for atomic kinetics / rad transport are largely independent of small-scale features + no Courant condition
  - isolated high-Z mix (e.g. from fill tube) may be an exception

Mesh modifications should be made <u>before</u> constructing the xfile

Options when running Cretin

Modifying elements and/or number densities:

- Source commands connect Cretin elements to xfile number densities

  source ni *ix iz z [a] [multiplier] [ireg]*

  iz identifies Cretin element defined with "*atoms*" command

  z [a] identifies number density section in xfile

  multiplier [ireg] changes number densities [in region *ireg*]

- Multiple source commands can refer to each xfile number density section

Producing spectral opacities and emissivities:

- Produced only if a spectral frequency mesh is present
- **switch(53) = -1** : produce spectral opacities without doing spectral transport
- **"dump drat"** : sends mesh, spectral opacities, etc. to dump files *xxx.d00, .d01, …*

Element substitutions are particularly useful when designing spectroscopy experiments

Postprocessing setup

Separate xfiles for each time interval (or timestep) if steady-state kinetics

1. yorick -i do_xfile.i — Produce ASCII xfile from rad-hydro output
2. cretin thd.xfa source — (Optional) produce PDB xfile
3. cretin thd.gen — Calculate populations with kinetics + radiation transport
4. cretin thd.r00 thd.gen **spectra** define SPECTRUM -o spec — Calculate spectral opacities and emissivities
5. yorick -i get_spec.i
   – or –
   cretin spec.d00 thd.gen **drat** define DRAT -o thd_spec — Calculate detector spectra

Splitting the calculation into parts allows changing the spectral frequency mesh or detector specifications without redoing earlier parts

Cretin postprocessing example

A single generator with #ifdef / #endif sections can do all 3 parts

```
#ifdef 1D
  geometry sphere
#else
  geometry rz
#endif

xfile  thd.xdf

switch  25    0        ! steady-state kinetics
switch  28    1        ! steady-state initialization
switch  29    0        ! use xfile timesteps
switch  44   10        ! max iterations per timestep
switch 120    1        ! show convergence diagnostics

#ifndef 1D
  switch  53  -1       ! no spectral transfer, opacities only
#endif
```

```
#ifdef SPECTRUM

spectrum  75    5000.   7500.
spectrum 300    7500.   9000.  1.    ! Cu K-shell
spectrum 800    9000. 13000.  1.    ! Ge K-shell
spectrum 100   13000. 20000.

#ifdef 1D
  plot
    xvar  sp_energy
    yvar  jsparea        0  -1
#else
  dump drat
#endif

#endif


#ifdef DRAT
#include  drat.gen
#endif
```

Postprocessing example – element substitution

xfile:

```
znuc  8
   1.00000    1.00000    1.00000    6.00000    1.00000    8.00000    32.0000    2.00000
anuc  8
   1.00000    2.00000    3.00000    0.00000    0.00000    0.00000    0.00000    0.00000
```

generator:

```
atoms  iz=1   dca_01   h
atoms  iz=2   dca_02   he
atoms  iz=6   dca_06   c
atoms  iz=8   dca_08   o

#ifdef GE
atoms  iz=32  dca_32k  ge
#endif

#ifdef CU
atoms  iz=29  dca_29k  cu
#endif

#ifdef SI
atoms  iz=14  dca_14   si
atoms  iz=32  dca_32k  ge
#endif
```

```
source  ni 1   1   1
source  ni 1   2   2
source  ni 1   6   6
source  ni 1   8   8

#ifdef GE
source  ni 1  32  32 0
#endif

#ifdef CU
source  ni 1  29  32 0  CUMULT
#endif

#ifdef SI
source  ni 1  14  32 0  SIMULT1   4
source  ni 1  14  32 0  SIMULT2   5
source  ni 1  32  32 0  GEMULT2  5
source  ni 1  32  32 0  GEMULT1  6
#endif
```

```
alias  GEMULT1  1.
alias  GEMULT2   0.5
alias  CUMULT    1.4
alias  SIMULT1   2.
alias  SIMULT2   1.
```

Substitute Cu for Ge @ 1.4x

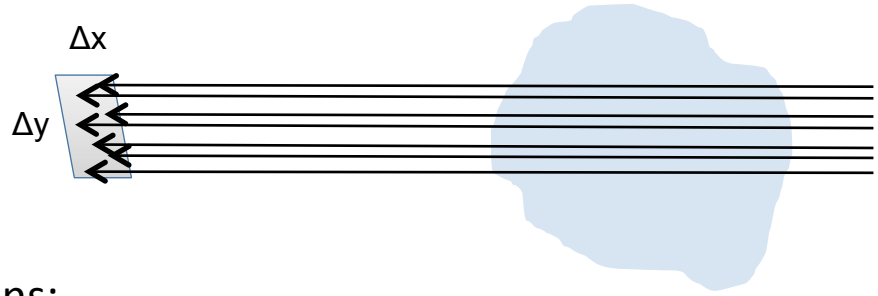
Substitute Si for Ge in region 4 @ 2x
+ partial substitution in region 5

Calculating detector spectra

Intensities reaching a detector are obtained from formal radiation transport along the line-of-sight to the detector

→ ray trace through the mesh

$$I_\nu = \int_0^s e^{-\alpha_\nu s'} \eta_\nu ds'$$



Ray tracing options:

1. Yorick running DRAT (2D) or HEX (3D)
   ICF program has extensive experience with DRAT + processing scripts
   drat_d00.i runs DRAT on Cretin dump files for 1D / 2D
2. Cretin "drat" option – available in 1D / 2D / 3D
   same functionality as DRAT/HEX plus Doppler shifting
   provides additional information along rays
   output is not compatible with DRAT processing scripts (but could be)
3. VisIt – not connected yet (any volunteers?)

Detector spectrum example

DRAT:

```
yorick -i drat_d00.i

> drat_d00, "spec.d00", 100, , 0.01
> save_d00, "thd.spec"
```

Read mesh and opacities from *thd.d00*
Detector plane extends ±100 μm in each direction
w/ 100 pixels in each (half-) direction
Equatorial + polar views aimed at (0.,0.)
Save results in PDB file *thd.spec*

Cretin drat option:

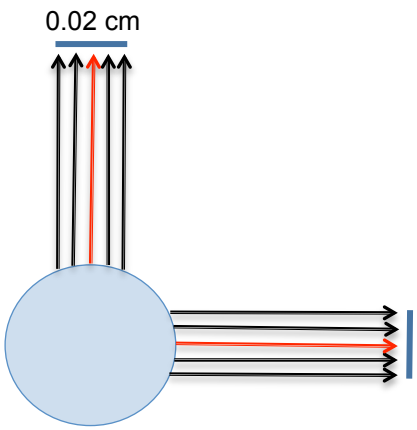
```
cretin spec.d00 drat.gen drat -o thd_spec

alias  MU0   1.          alias  DX     0.01
alias  PHI0  PI/2.       alias  XMIN   0.
alias  R0    0.          alias  YMIN   -DX
alias  Z0    1.          alias  NPX0   51
                         alias  NPY0   51
alias  MU1   0.
alias  PHI1  PI/2.       alias  NPX    101
alias  R1    1.          alias  NPY    101
alias  Z1    0.

drat  MU0  PHI0  R0 Z0 0.     0.   DX YMIN DX NPX0 NPY0  [doppler]
drat  MU1  PHI1  R1 Z1 0.  XMIN DX YMIN DX NPX   NPY    [doppler]
```

Results will be in PDB file *thd_spec.drt*

Pixel mesh is node-centered



Polar view
Equatorial view

Central rays are defined by a direction (μ,ϕ) and a reference point (x,y,z) or (r,$z,0$) <u>outside</u> the mesh