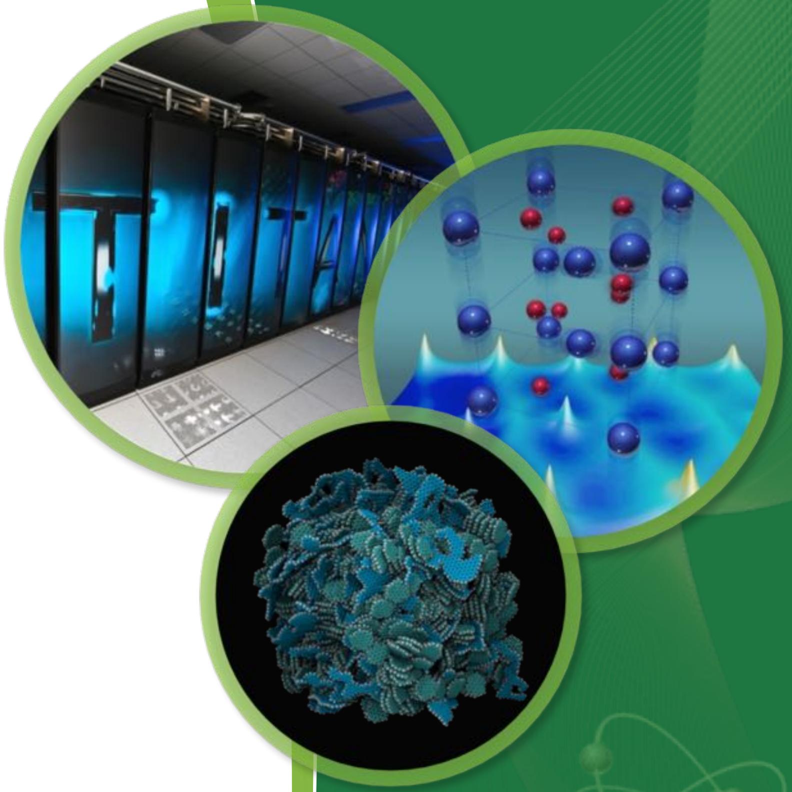
CAAR Porting Experience: FLASH

J. Austin Harris

Scientific Computing Group
Oak Ridge National Laboratory

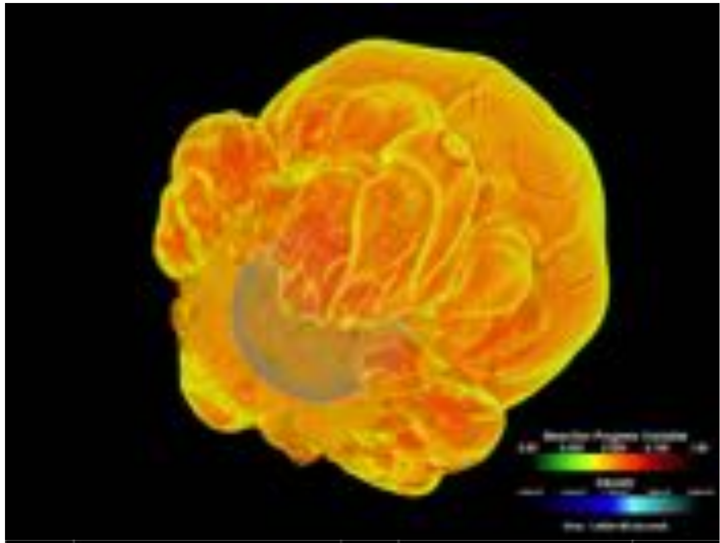


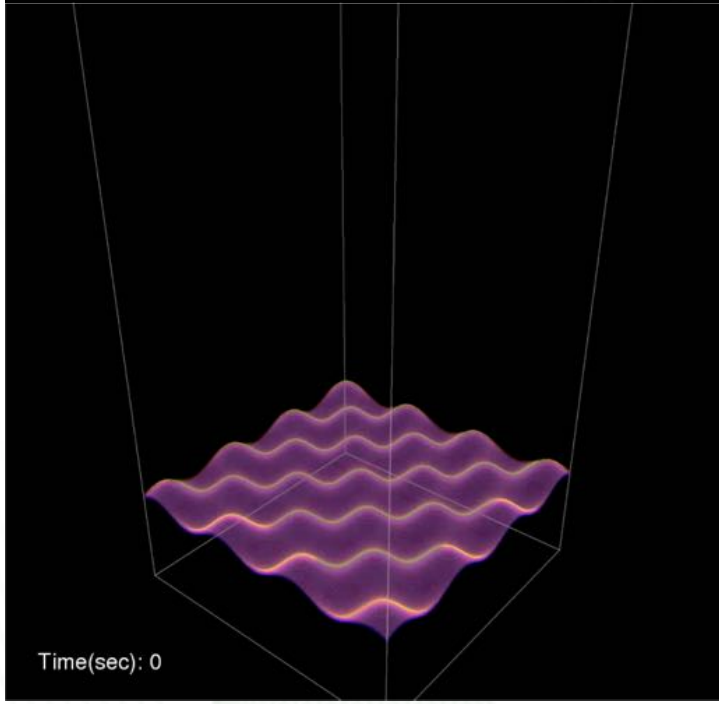
ORNL is managed by UT-Battelle
for the US Department of Energy

OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY

FLASH code

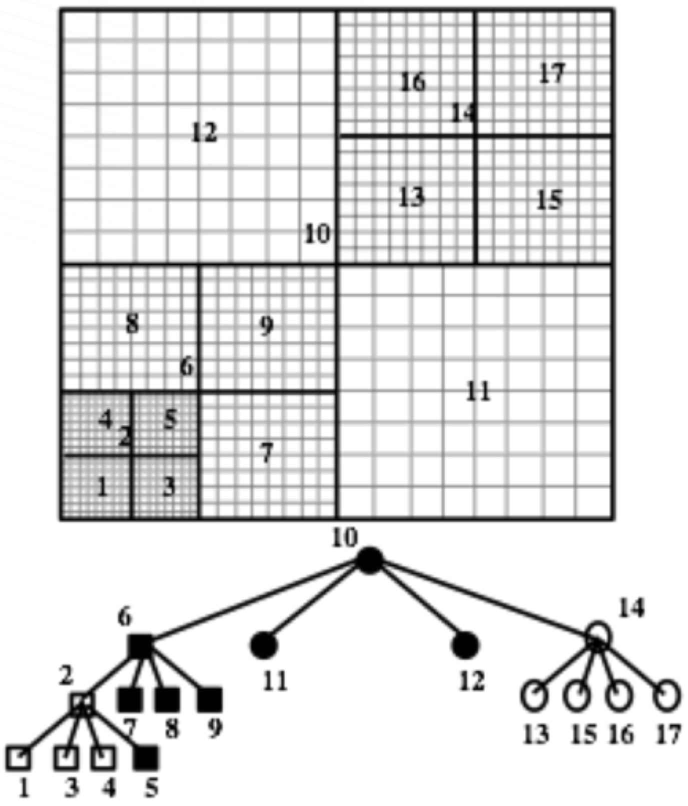
- Component-based, MPI+OpenMP parallel, adaptive mesh refinement (AMR) code supporting:
  - Directionally unsplit hydrodynamics
  - Multipole gravity solver
  - Nuclear burning network
  - Stellar equation of state
  - Turbulent flame interaction model
- The code has been used to simulate a variety of phenomena:
  - thermonuclear and core-collapse supernovae
  - galaxy cluster formation
  - classical novae
  - formation of proto-planetary disks
  - high-energy-density physics

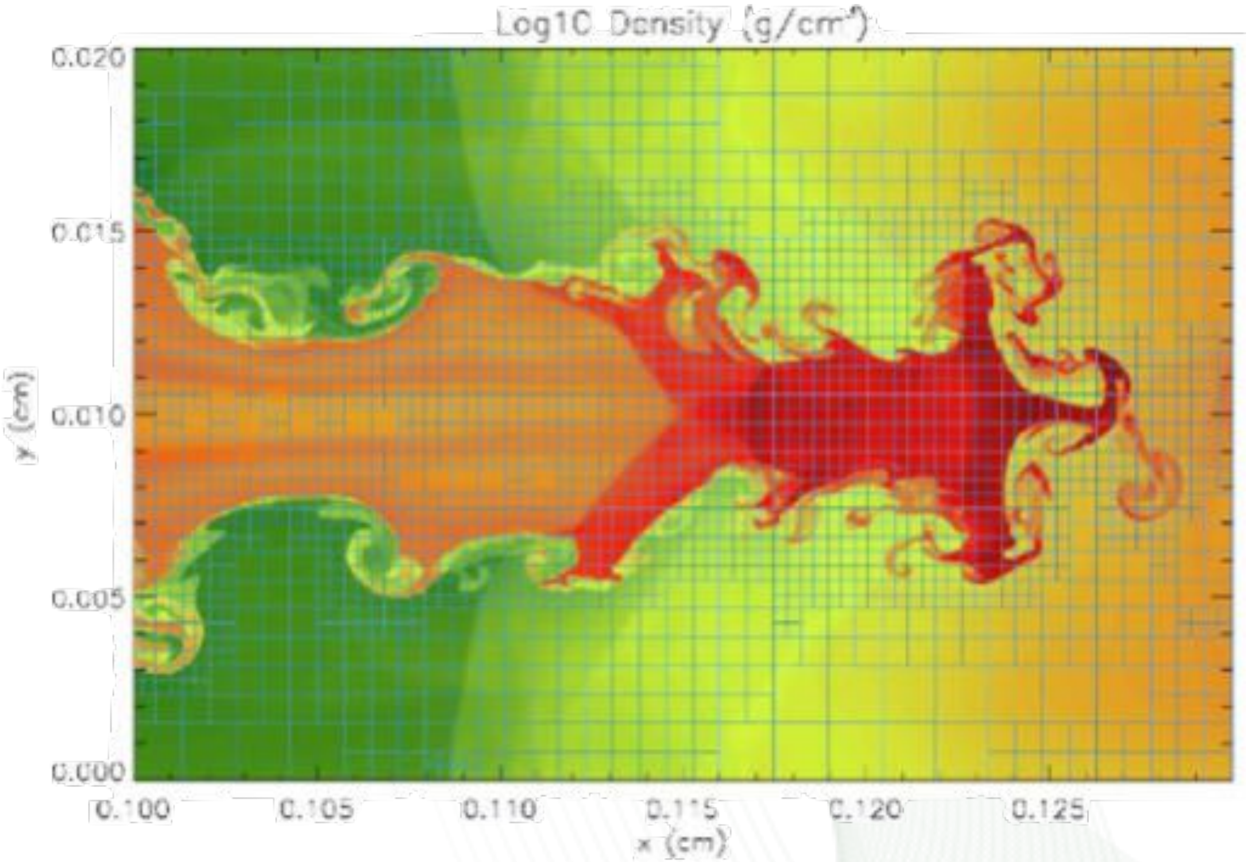




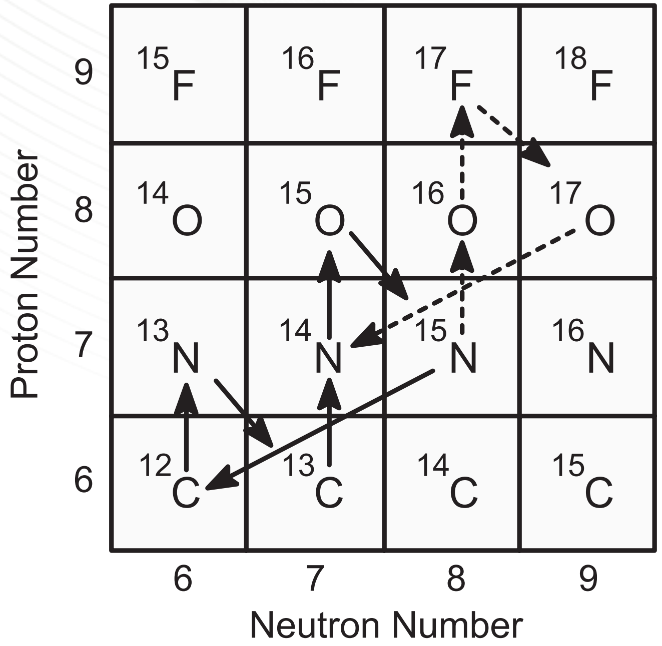
FLASH AMR

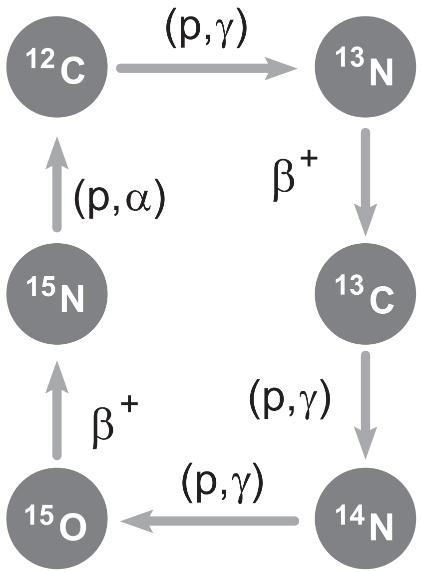
- Currently uses octree-based PARAMESH (MacNeice+, 2000)
- Moving to ECP-supported AMREx





Nuclear kinetics





Main cycle

- Reaction network described by:

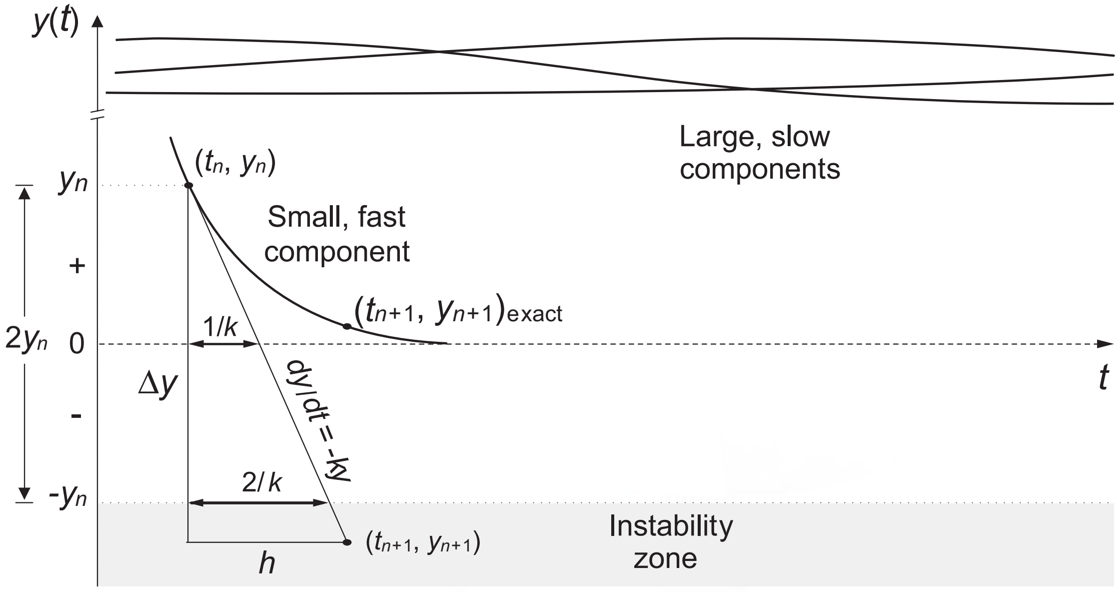
$$f(Y_i) = \frac{dY_i}{dt} = \sum_j \mathcal{N}^i{}_j \lambda_j Y_j + \sum_{j,k} \mathcal{N}^i{}_{j,k} \rho N_A \langle \sigma v \rangle_{j,k} Y_j Y_k + \sum_{j,k,l} \mathcal{N}^i{}_{j,k,l} \rho^2 {N_A}^2 \langle \sigma v \rangle_{j,k,l} Y_j Y_k Y_l + \cdots$$

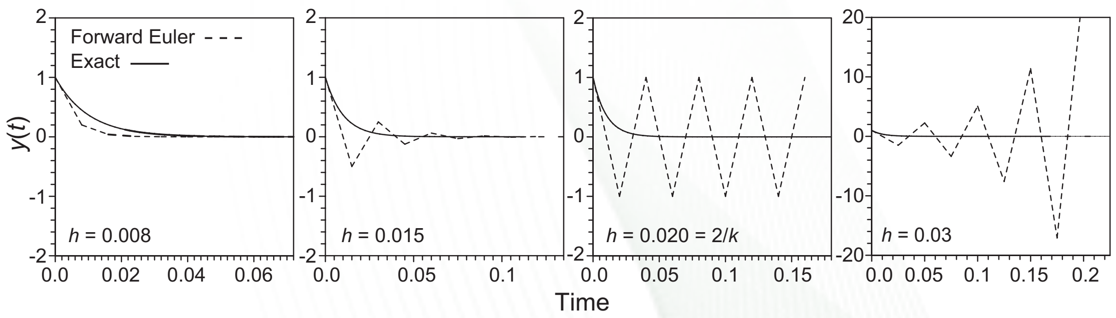
A digression about stiffness

- Practically, stiff if stepsize h set by numerical stability from large variations in timescales
- Formally, system of equations stiff if Jacobian obeys

$$\mathcal{S} = \frac{\max|\Re(\lambda_j)|}{\min|\Re(\lambda_j)|} \gg 1$$

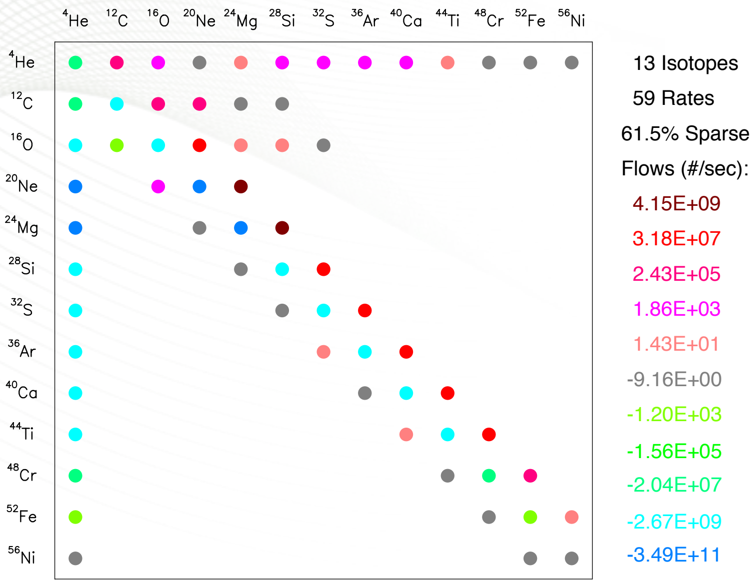
- $\mathcal{S} > 10^{15}$ not uncommon in astrophysics
- Two approaches:
    1. Remove stiffness, relax timestep constraints
    2. (Semi-)implicit numerical integration

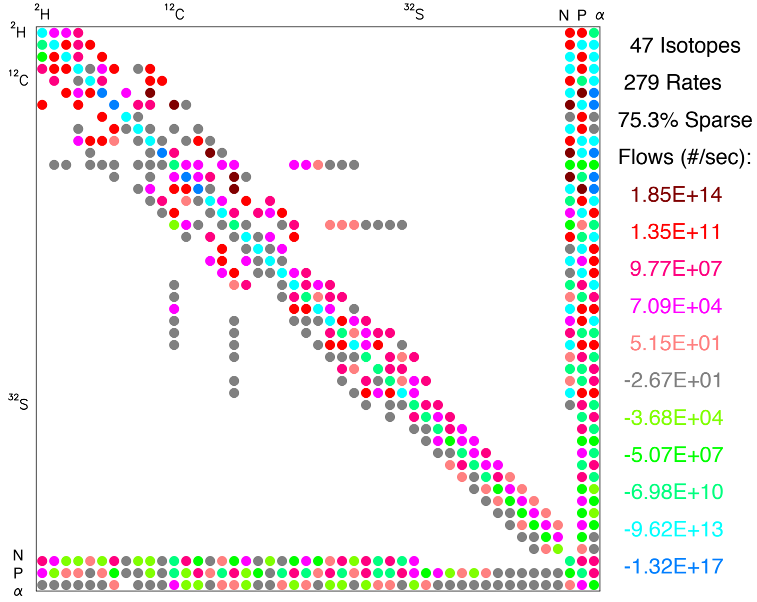


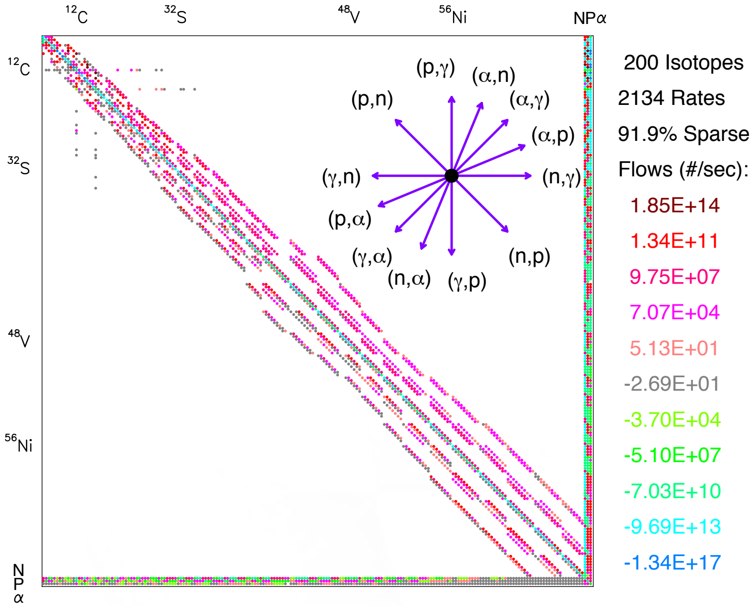


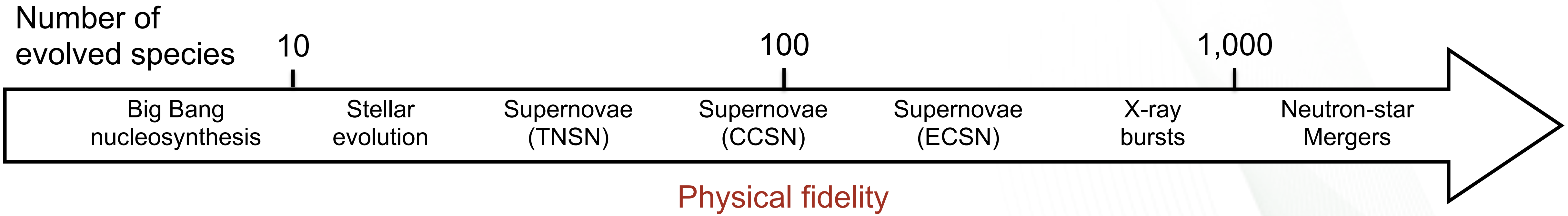
OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY

Reaction networks

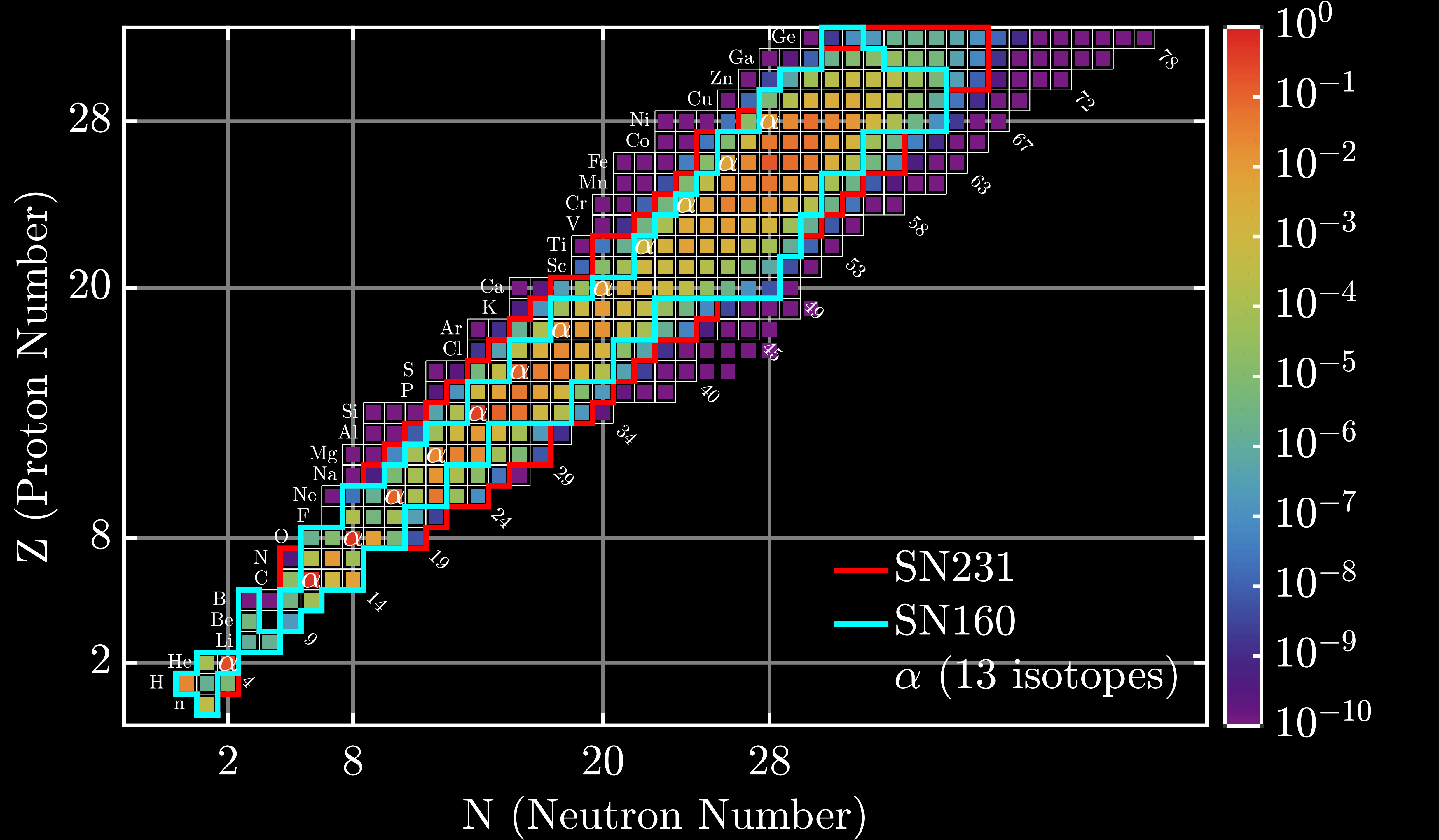








Physical fidelity
FLOPS
Sparsity



Z (Proton Number)
N (Neutron Number)
SN231
SN160
α (13 isotopes)
2
8
20
28
n
H
He
Li
Be
B
C
N
O
F
Ne
Na
Mg
Al
Si
P
S
Cl
Ar
K
Ca
Sc
Ti
V
Cr
Mn
Fe
Co
Ni
Cu
Zn
Ga
Ge
4
9
14
19
24
29
34
40
45
49
53
58
63
67
72
78
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
10^{-9}
10^{-10}

XNet

- Stand-alone code for evolving arbitrary reaction networks
- Originally designed to independently evolve particles/zones (SPMD)
  - Minimal overhead for CPU shared-memory parallelism
  - Unfortunately, not ideal for GPU

- FLASH CAAR project:
  - Replace small "hard-wired" reaction network in FLASH with XNet interface and use programming model centered around GPU-optimized libraries

XNet

- Implicit solver for stiff system of ODEs
  - Backward Euler (first-order):

$$(\mathbf{I} - h\mathbf{J})\Delta\vec{Y}^{[m]} = hf\left(\vec{Y}_{n+1}^{m}\right) + \vec{Y}_{n+1}^{0} - \vec{Y}_{n+1}^{m}$$

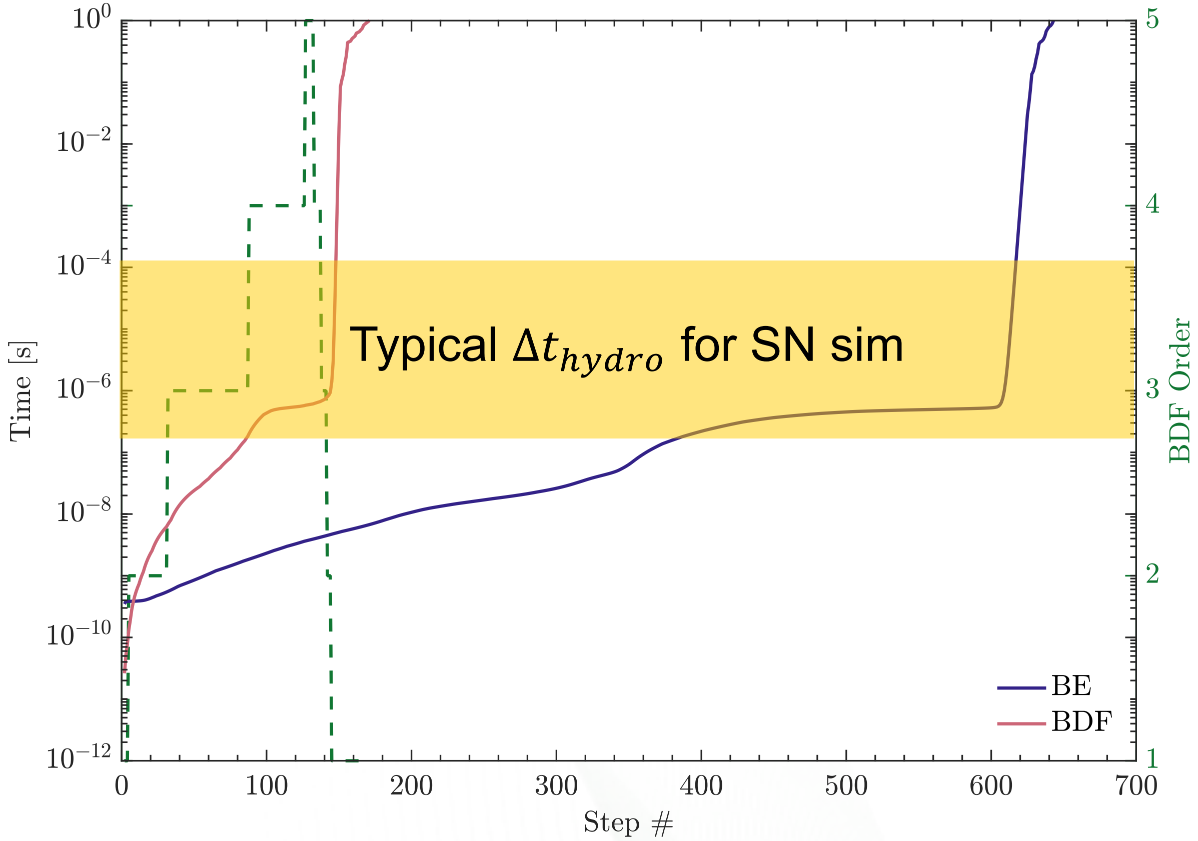
  - Added variable-order BDF (Gear+ 1971):
    - Motivated by results of Longland+ (2014)
    - Idea: Use past behavior to predict solution
    - Predictor step:

$$\mathbf{z}_n = \left[\vec{Y}_n, h\dot{\vec{Y}}_n, h^2\ddot{\vec{Y}}_n, \ldots, \frac{h^q\vec{Y}_n^{(q)}}{q!}\right],$$

$$\mathbf{z}_{n+1}^{0} = \mathbf{z}_n\mathbf{A}(q), \qquad \mathbf{A}(q) \equiv \text{Pascal matrix}$$

    - Corrector step:

$$\left(\mathbf{I} - \frac{h}{l_1}\mathbf{J}\right)\Delta\vec{Y}^{[m]} = \frac{h}{l_1}\left[f\left(\vec{Y}_{n+1}^{0}\right) - f\left(\vec{Y}_{n+1}^{m}\right)\right] + \vec{Y}_{n+1}^{0} - \vec{Y}_{n+1}^{m}$$



Fewer number of steps to converge
- Faster time to solution
- Better load balancing

XNet in FLASH

- FLASH burner restructured to operate on multiple zones at once from all local AMR blocks for XNet to evolve simultaneously

```
!$omp parallel shared(…) private(…)
!$omp do
do k = 1, num_zones
  do j = 1, num_timesteps
    <build linear system>
    dgetrf(…)
    dgetrs(…)
    <check convergence>
  end do
end do
!$omp end do
!$omp end parallel
```

```
!$omp parallel shared(…) private(…)
!$omp do
do k = 1, num_local_batches
  do j = 1, num_timesteps
    <CPU operations>
    !$acc parallel loop
    do ib = 1, batch_size
      <build ib'th linear system>
    end do
    !$acc end parallel loop
    <send system to GPU>
    cublasDgetrfBatched(…)
    cublasDgetrsBatched(…)
    <send results to CPU>
    <check convergence>
  end do
end do
!$omp end do
!$omp end parallel
```

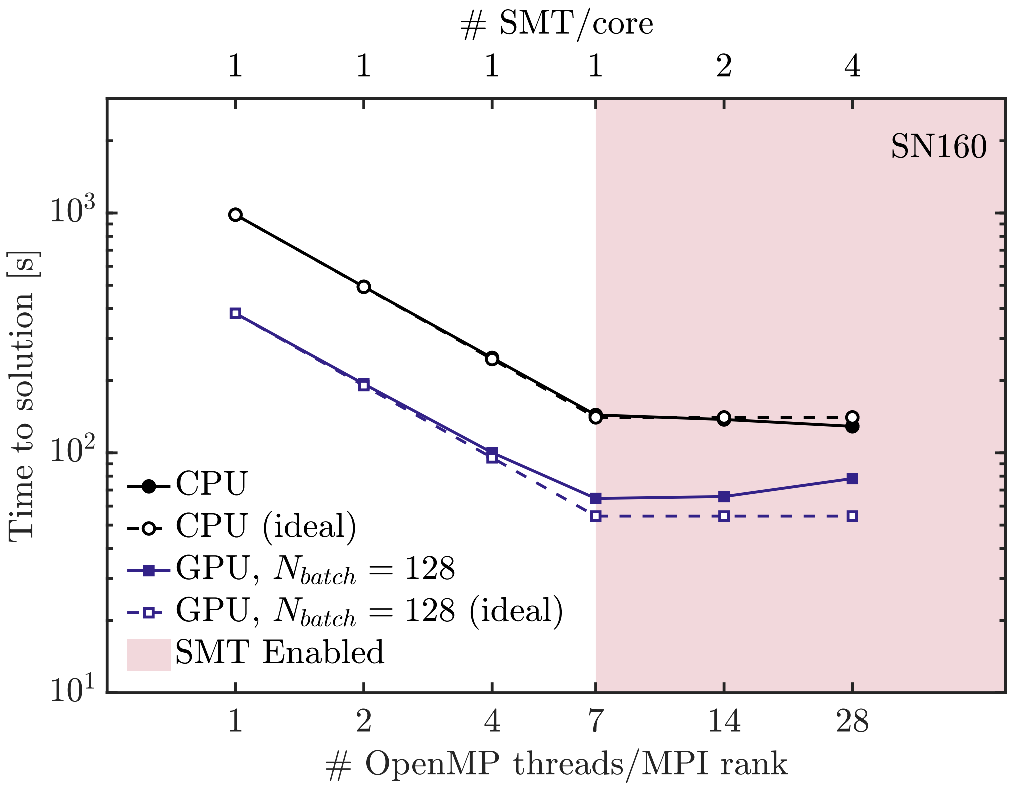
XNet in FLASH

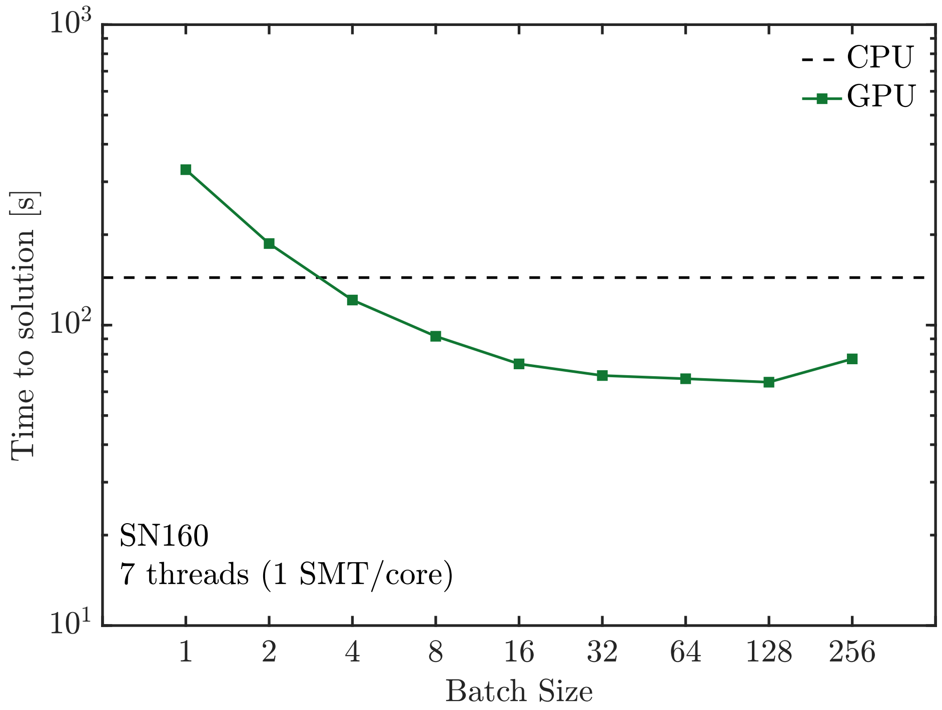
- Fortran data structures

```
Real(dp), Pointer                  :: jac(:,:,:) ! CPU data (pointers for pinned memory)
Type(C_PTR)                        :: hjac ! C pointers for pinned memory
Type(C_PTR)                        :: djac ! Device pointers for arrays
Integer(C_INTPTR_T), Pointer       :: djacf(:,:,:)
Type(C_PTR), Allocatable, Target   :: djaci(:) ! Arrays of pointers to each device array batch element address
Type(C_PTR)                        :: hdjac_array, djac_array ! Host and device addresses for the arrays of device pointers

! Allocate CPU memory (pinned for asynchronous host<->device copy)
istat = cudaHostAlloc(hjac, sizeof_jac, cudaHostAllocDefault)
Call c_f_pointer(hjac, jac, (/msize,msize,nzbatchmx/))

istat = cudaMalloc(djac, msize*msize*nzbatchmx*sizeof_double) ! Allocate GPU memory for arrays
Call c_f_pointer(djac, djacf, (/msize,msize,nzbatchmx/)) ! Setup fortran pointers to device-addresses of device arrays

! Setup arrays of pointers to each device array batch element address
Allocate (djaci(nzbatchmx))
do i = 1, nzbatchmx
  djaci(i) = c_loc(djacf(1,1,i)) ! Get the device-addresses for this batch element
end do
hdjac_array = c_loc(djaci(1)) ! Get the host-addresses for the arrays of device-pointers

! Allocate GPU memory for batched GPU operations and copy array of pointers to device
istat = cudaMalloc(djac_array, nzbatchmx*sizeof_cptr)

...

! Copy the system to the GPU
istat = cublasSetMatrixAsync(msize, msize*nzbatchmx, sizeof_double, hjac, msize, djac, msize, stream)
```

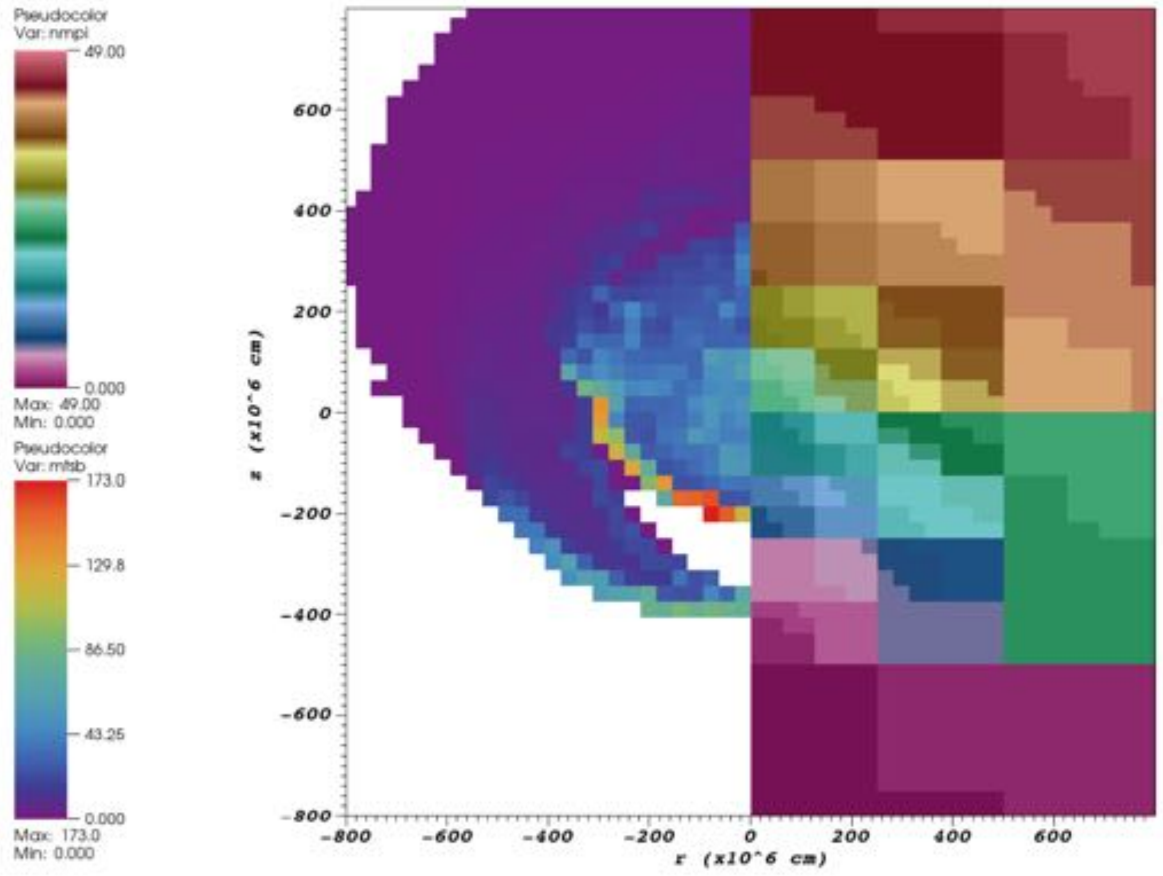
FLASH Performance w/ Xnet
Single node tuning

- Tests performed on single Summit Phase I node
  - 2 IBM Power9 (22 cores each), 6 NVIDIA “Volta” V100 GPUS
- 1 3D block (16^3 = 4096 zones) per rank per GPU evolved for 20 FLASH timesteps

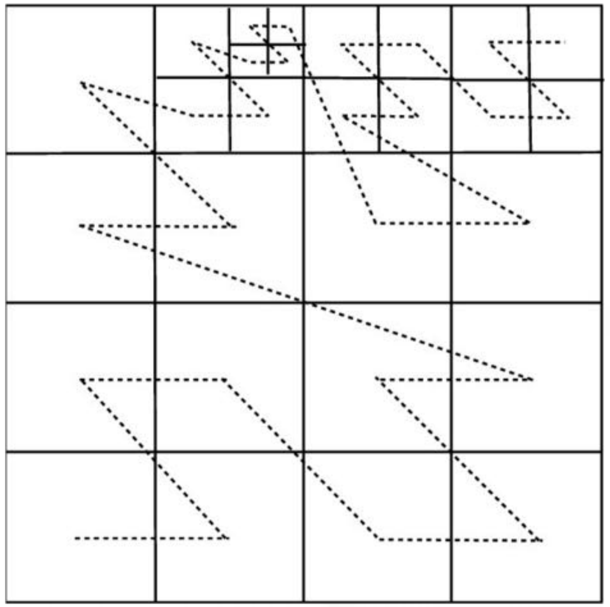




FLASH AMR Optimization

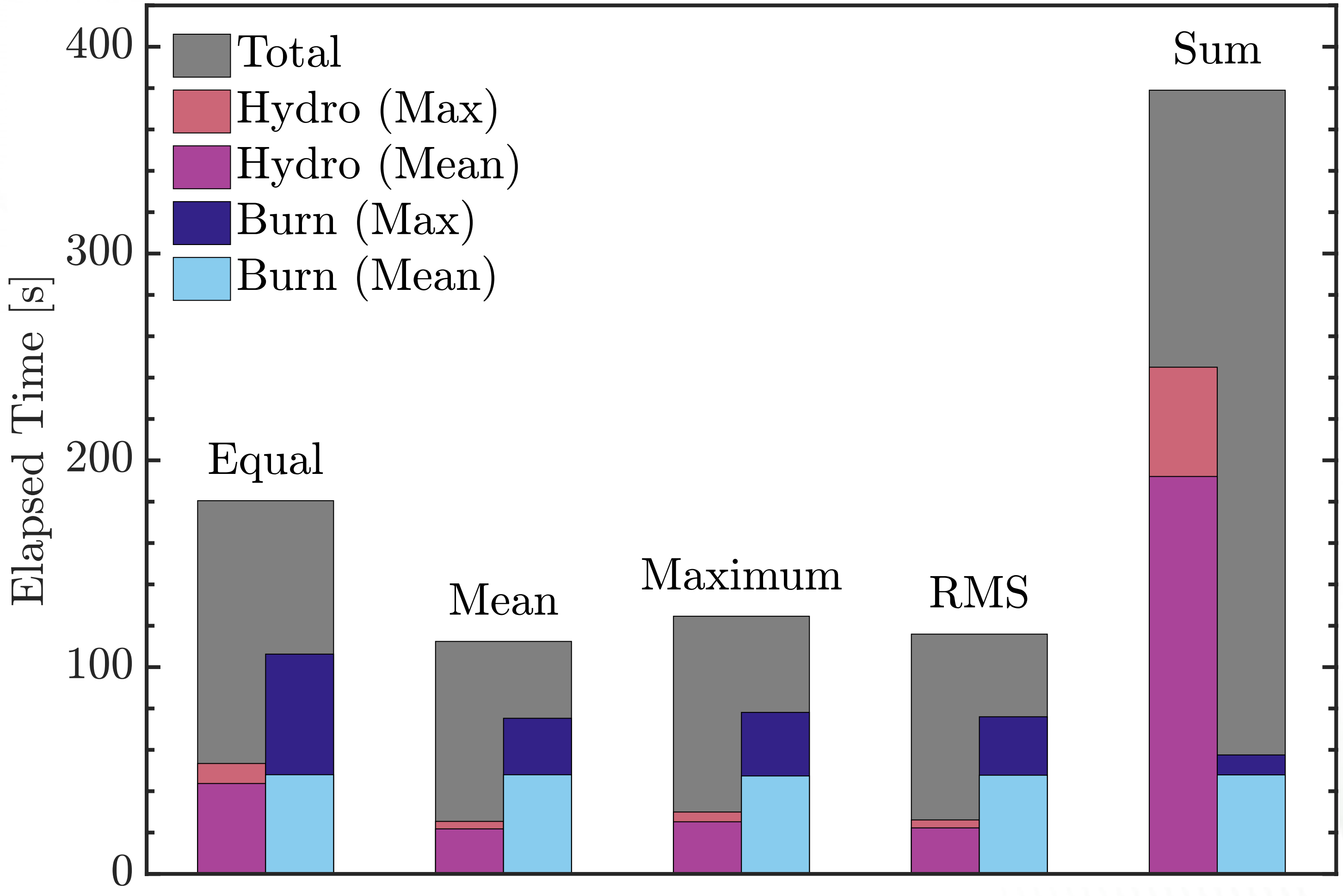


- Problem: Computational load can be quite unevenly distributed



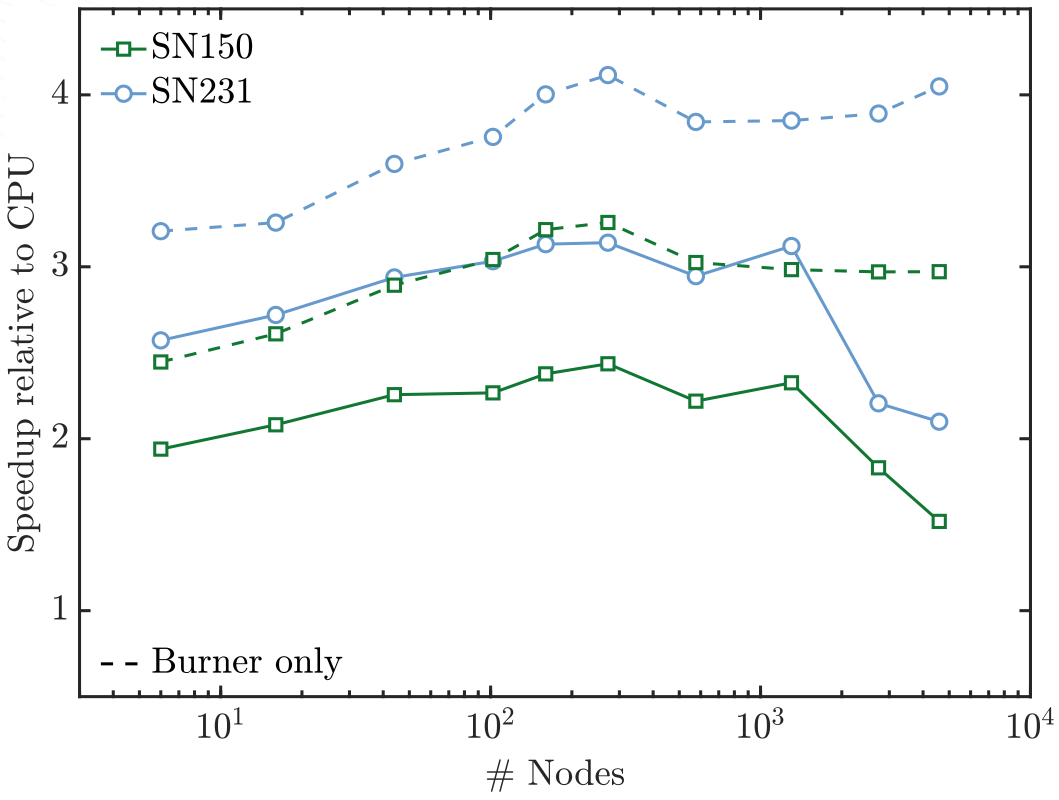
- Solution: Weight the Morton space-filling curve average number of burning steps per block
  - ~1.5x speedup at scale

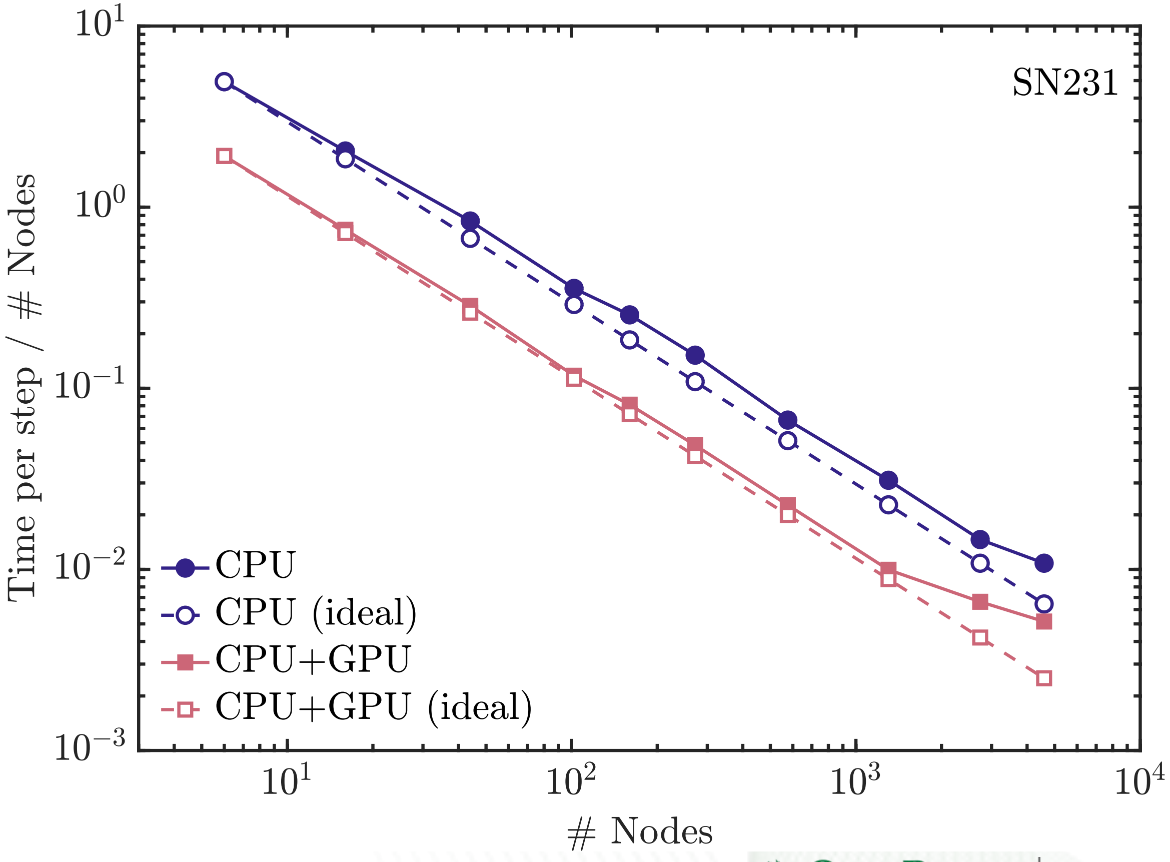
FLASH AMR Optimization



FLASH scaling results

- Real physics problem:
  - Centrally detonated white dwarf with 231 species
  - Nearly ideal weak scaling to ~1000 nodes





OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY

Other CAAR Work

- Other FLASH developments for CAAR:
  - GPU equation of state (Tom Papatheodore)
  - Non-blocking communications in gravity solver (Hannah Klion)

Questions?