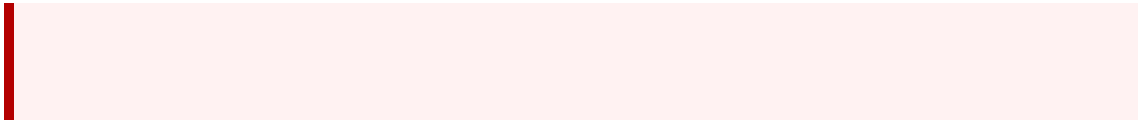
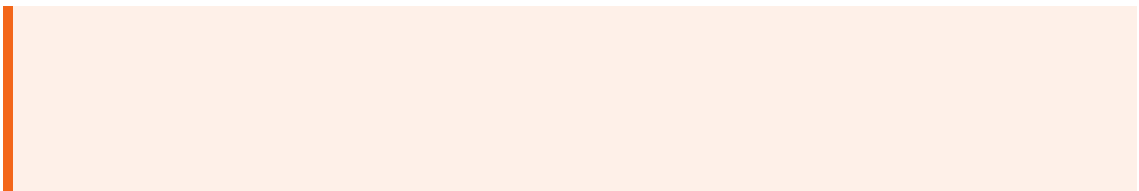
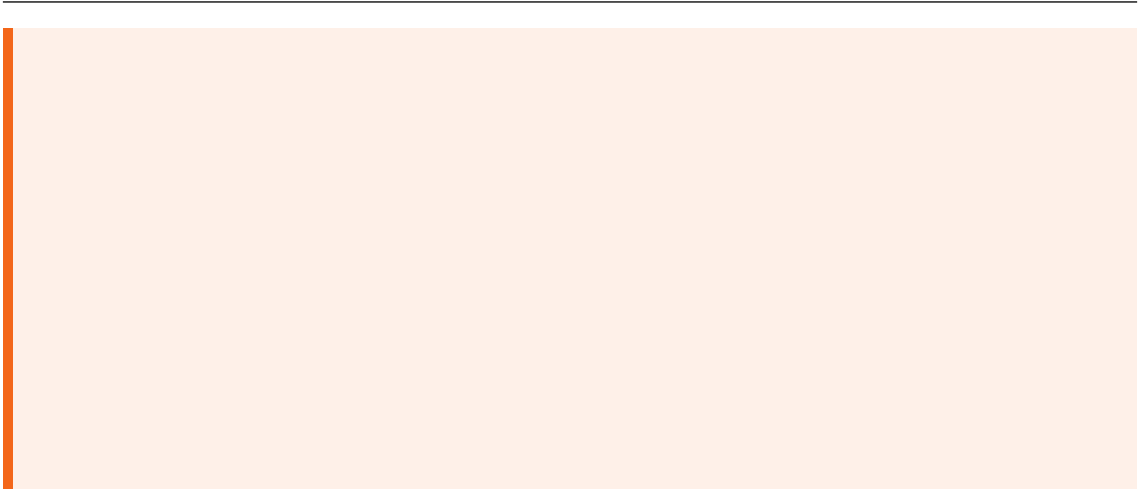
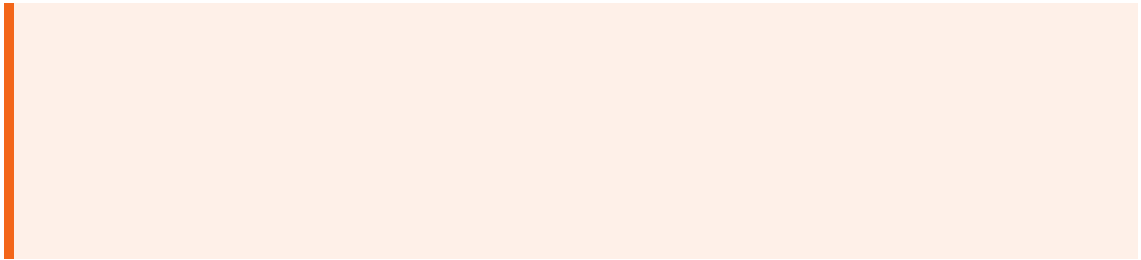
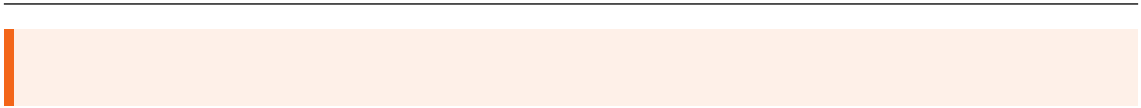

RPacket

trace_packet







```
# CPU build (serial)
make

# CPU build with OpenMP parallelism
make OMP=1

# CUDA GPU build
make cuda
```

```
make clean
```

```
kurucz_cd23_chianti_H_He.h5
```

```
math.hstdio.hstdlib.hstring.h
```

```
# CPU: reference model with 200K packets, 20 iterations
./lumina tardis_reference 200000 20

# CUDA: same with GPU acceleration
./lumina_cuda atomic/kurucz_cd23_chianti_H_He.h5 200000 output.
csv
```

RPacket

RPacket

```
typedef struct {
    double r;                // radial position [cm]
    double mu;               // cos(theta) direction
    double nu;               // lab-frame frequency [Hz]
    double energy;           // packet energy [erg]
    int current_shell_id;    // shell index [0..n_shells-1]
    int next_line_id;        // Sobolev sweep bookmark
    PacketStatus status;     // IN_PROCESS / EMITTED /
                            REABSORBED
    int index;               // packet ID (for RNG seeding)
} RPacket;
```

Geometry

```
typedef struct {
    int n_shells;            // number of shells (default: 30)
    double *r_inner;         // [n_shells] inner radii [cm]
    double *r_outer;         // [n_shells] outer radii [cm]
    double *v_inner;         // [n_shells] inner velocities [cm/
                            s]
    double *v_outer;         // [n_shells] outer velocities [cm/
                            s]
    double time_explosion;    // t_exp [seconds]
} Geometry;
```

OpacityState

```
typedef struct {
    int n_lines, n_shells;
    double *line_list_nu;           // [n_lines] rest frequencies,
        descending
    double *tau_sobolev;             // [n_lines * n_shells]
    double *electron_density;        // [n_shells] n_e [cm^-3]
    double *t_electrons;             // [n_shells] T_e [K]

    // Macro-atom transition data
    int n_macro_levels, n_macro_transitions;
    int *macro_block_references;     // [n_levels+1]
    int *transition_type;            // [n_transitions]
    int *destination_level_id;       // [n_transitions]
    int *transition_line_id;         // [n_transitions]
    double *transition_probabilities; // [n_transitions *
        n_shells]
    int *line2macro_level_upper;     // [n_lines]
} OpacityState;
```

PlasmaState

```
typedef struct {
    int n_shells;
    double *W;                      // [n_shells] dilution factor
    double *T_rad;                   // [n_shells] radiation temperature
        [K]
    double *rho;                     // [n_shells] mass density [g/cm^3]
    double *n_electron;              // [n_shells] electron density [cm
        ^-3]
    double T_e_T_rad_ratio;          // default: 0.9
} PlasmaState;
```

AtomicData

```
typedef struct {
    // Per-line data
    int *line_atomic_number;         // [n_lines] Z
    int *line_ion_number;            // [n_lines] ionization stage
    double *line_f_lu;               // [n_lines] oscillator
        strength
    double *line_wavelength_cm;      // [n_lines] rest wavelength

    // Energy levels
    int n_levels;
    double *level_energy_eV;         // [n_levels]
```

```

int      *level_g;                // [n_levels] statistical
weight
int      *level_metastable;      // [n_levels] 0 or 1

// Ionization energies
int      n_ionization;
double   *ioniz_energy_eV;       // [n_ionization]

// Zeta correction factors (dilute non-LTE)
double   *zeta_data;             // [n_zeta_ions *
    n_zeta_temps]

// Abundances
double   *abundances;            // [n_elements * n_shells]

// Computed quantities (updated each iteration)
double   *ion_number_density;    // [n_ion_pops * n_shells]
double   *partition_functions;   // [n_ion_pops * n_shells]
} AtomicData;

```

Estimators

```

typedef struct {
    double *j_estimator;          // [n_shells] integral of E*ds
    double *nu_bar_estimator;     // [n_shells] integral of E*nu*
    ds
    double *j_blue_estimator;     // [n_lines * n_shells] (CPU
    only)
    double *Edotlu_estimator;     // [n_lines * n_shells] (CPU
    only)
} Estimators;

```

j_blueEdotluatomicAdd

lumina_transport.c
trace_packet

```
double tau_trace_combined = 0.0;
for (int j = pkt->next_line_id; j < n_lines; j++) {
    double nu_line = line_list_nu[j];
    double d_line = compute_d_line(pkt, nu_line);

    if (d_line < 0 || d_line > d_boundary) break;

    double tau_line = tau_sobolev[j * n_shells + shell_id];
    tau_trace_combined += tau_line;

    if (tau_trace_combined > tau_event) {
        // Line interaction!
        *interaction_type = INTERACTION_LINE;
        *d_min = d_line;
        pkt->next_line_id = j;
        return;
    }
}
```

lumina_plasma.c

`\x93NUMPY`

`shapedtypelfortran_order`

`macro_atom_data.csv,atomic_number,...strtok()`

```
int threads_per_block = 256;
int blocks = (n_packets + threads_per_block - 1)
             / threads_per_block;
// Max blocks: 131072 (was 1024 -- critical bug #13)
transport_kernel<<<blocks, threads_per_block>>>(...);
```


```
CUDA_MAX_BLOCKS = 1024
```

```
uint64
```

```
__device__ void init_rng(uint64_t *state, uint64_t seed) {
    // SplitMix64 to expand seed into 4 state words
    state[0] = splitmix64(&seed);
    state[1] = splitmix64(&seed);
    state[2] = splitmix64(&seed);
    state[3] = splitmix64(&seed);
}
```

atomicAdd

```
__device__ void update_estimators(  
    double *d_j_est, double *d_nu_bar_est,  
    int shell_id, double comov_energy,  
    double comov_nu, double distance)  
{  
    atomicAdd(&d_j_est[shell_id],  
              comov_energy * distance);  
    atomicAdd(&d_nu_bar_est[shell_id],  
              comov_energy * distance * comov_nu);  
}
```



__shared__ ShellCache
cuda_reset_estimators()

```
# Clone the repository
git clone git@github.com:kjhan0606/lumina-sn.git
cd lumina-sn

# Build CPU version
make

# (Optional) Build GPU version
make cuda

# Verify with a quick test (1000 packets, 5 iterations)
./lumina tardis_reference 1000 5

# Production run (200K packets, 20 iterations)
./lumina tardis_reference 200000 20

# Check output
head lumina_spectrum.csv
```

```
data/tardis_reference/
```

```
data/tardis_reference/
```

```
config.json           # Simulation parameters
geometry.csv          # Shell radii and velocities
density.csv           # Mass density per shell
abundances.csv        # Element mass fractions
electron_densities.csv # Initial electron densities
plasma_state.csv      # Initial W, T_rad per shell
line_list.csv         # Atomic line data
tau_sobolev.npy       # Reference optical depths
transition_probabilities.npy
macro_atom_data.csv
macro_atom_references.csv
line2macro_level_upper.npy
levels.csv            # Energy levels
ionization_energies.csv
zeta_ions.csv
zeta_temps.csv
zeta_data.npy
atom_masses.csv
```

`scripts/fit_parameter_search.py`
