LUMINA-SN

Monte Carlo Radiative Transfer for Type Ia Supernovae

A Complete Technical Manual

K. J. Han

February 2026

LUMINA-SN: Luminosity-driven Monte Carlo Supernova Spectral Synthesizer

This code implements 1D Monte Carlo radiative transfer in homologously expanding supernova ejecta, following the formalism of Lucy [3–5] and the TARDIS code [2]. Written in C99 with CUDA GPU acceleration.

Version 2.0 — February 2026

Contents

I

Physics & Theory

1 Introduction to Type Ia Supernovae

1.1 What is a Type Ia Supernova?

A Type Ia supernova (SN Ia) is the thermonuclear explosion of a carbon-oxygen white dwarf star that has reached a critical mass near the Chandrasekhar limit ($M_{\rm Ch} \approx 1.4\,M_\odot$). The explosion completely unbinds the star, leaving no compact remnant, and synthesizes approximately $0.6\,M_\odot$ of radioactive ^{56}Ni [7].

Important: SN Ia are cosmological standard candles: their peak luminosity correlates with their light-curve decline rate (the Phillips relation, [8]), enabling precise distance measurements to galaxies. This led to the discovery of the accelerating expansion of the Universe.

1.2 The Expanding Ejecta

After the explosion, the ejecta expand freely into a vacuum. Within hours, the expansion reaches a state of **homologous expansion**:

$$\boxed{v(r,t) = \frac{r}{t}} \tag{1.1}$$

where r is the radial distance from the center and t is the time since explosion. This means velocity maps directly to radius: faster material is farther out.

Definition 1.1 — Homologous Expansion. In homologous expansion, each fluid element moves at constant velocity. The density at any velocity coordinate v evolves as:

$$\rho(v,t) = \rho_0(v)\left(\frac{t_0}{t}\right)^3 \tag{1.2}$$

where $\rho_0(v)$ is the density at reference epoch t_0, and the t^{-3} factor comes from the 3D volumetric dilution.

1.3 Spectral Features

SN Ia spectra are dominated by **P Cygni profiles**: blueshifted absorption troughs paired with redshifted emission peaks. These arise because:

1. Material approaching the observer (along the line of sight) absorbs photons at a blueshifted wavelength.
2. The surrounding envelope re-emits photons isotropically, producing a net emission component redward of the rest wavelength.

Table 1.1: Key spectral features in SN Ia near maximum light.

Ion	**Rest λ (Å)**	**Observed Range (Å)**	**Diagnostic Value**
Si II	6355	5800–6500	Expansion velocity, temperature
Si II	5972	5600–6000	Temperature indicator
S II	5454, 5640	5200–5700	"W" feature, burning completeness
Ca II	3934, 3968	3600–4000	H&K lines, high-velocity features
Ca II	8498, 8542, 8662	8000–8800	IR triplet
Fe II	4500–5200	4300–5200	Iron-group blanketing
O I	7774	7400–7900	Unburned oxygen indicator

1.4 The Inverse Problem

Given an observed spectrum, we want to determine the physical parameters of the explosion:

- Luminosity L and photospheric temperature T_{inner}
- Density profile $\rho(v)$ and its power-law exponents
- Chemical composition as a function of velocity (abundance tomography)
- Time since explosion t_{exp}

LUMINA-SN solves the *forward problem*: given these parameters, compute the emergent spectrum. Combined with Bayesian inference (Part III), this enables solving the inverse problem.

2 Radiative Transfer in Expanding Atmospheres

2.1 The Transfer Equation

The specific intensity I_ν along a ray satisfies:

$$\frac{dI_\nu}{ds} = -\kappa_\nu I_\nu + j_\nu \tag{2.1}$$

where s is the path length, κ_ν is the absorption coefficient (opacity), and j_ν is the emissivity.

In a supernova atmosphere, three opacity sources contribute:

1. **Electron scattering** (Thomson): frequency-independent, $\sigma_T = 6.652 \times 10^{-25}$ cm^2
2. **Line opacity** (Sobolev): resonant absorption in atomic transitions
3. **Continuum opacity**: bound–free and free–free (negligible in SN Ia, see §2.5)

2.2 The Sobolev Approximation

In homologous expansion, the velocity gradient $dv/dr = 1/t_{\rm exp}$ is constant. A photon sweeps through a line's resonance frequency over a very short distance (the *Sobolev length*). This means line interactions are *local*: each line either absorbs the photon or lets it pass.

Definition 2.1 — Sobolev Optical Depth. The optical depth of a line transition $l \to u$ in the Sobolev approximation is:

$$\boxed{\tau_{\rm Sob} = \frac{\pi e^2}{m_e c} f_{lu} \lambda_0 t_{\rm exp} n_l \left(1 - \frac{g_l n_u}{g_u n_l}\right)} \tag{2.2}$$

where f_{lu} is the oscillator strength, λ_0 is the rest wavelength, n_l and n_u are the lower and upper level populations, and g_l, g_u are the statistical weights.

The numerical coefficient is:

$$\frac{\pi e^2}{m_e c} = 0.02654 \text{ cm}^2 \text{ s}^{-1} \quad \text{(CGS)} \tag{2.3}$$

The *escape probability* from a Sobolev line is:

$$\beta_{\rm Sob} = \frac{1 - e^{-\tau_{\rm Sob}}}{\tau_{\rm Sob}} \tag{2.4}$$

2.3 The Dilute Radiation Field

Far from the photosphere, the radiation field is diluted. The mean intensity is:

$$J_\nu = W\, B_\nu(T_{\rm rad}) \tag{2.5}$$

where W is the **dilution factor** and $T_{\rm rad}$ is the **radiation temperature**. For a geometrically thin photosphere at radius $R_{\rm phot}$:

$$W(r) = \frac{1}{2}\left(1 - \sqrt{1 - \left(\frac{R_{\rm phot}}{r}\right)^2}\right) \tag{2.6}$$

Remark 2.1 — Physical Interpretation of W. At $r = R_{\rm phot}$: $W = 0.5$ (hemisphere illuminated). At $r \gg R_{\rm phot}$: $W \approx R_{\rm phot}^2/(4r^2) \to 0$ (point source). In practice, Monte Carlo estimators yield W values that include the effects of line scattering and fluorescence.

2.4 Frame Transformations

In homologous expansion, the comoving-frame (CMF) frequency differs from the lab-frame frequency:

$$\nu_{\rm cmf} = \nu_{\rm lab}\left(1 - \frac{\mu\, v}{c}\right) = \nu_{\rm lab}\left(1 - \frac{\mu\, r}{c\, t_{\rm exp}}\right) \tag{2.7}$$

where $\mu = \cos\theta$ is the direction cosine of the photon with respect to the radial direction, and $v = r/t_{\rm exp}$.

Important: As a photon propagates through a shell, the comoving-frame frequency changes *linearly* with distance s along the ray:

$$\nu_{\rm cmf}(s) = \nu_{\rm lab}\left(1 - \frac{r_0\,\mu_0 + s}{c\, t_{\rm exp}}\right) \tag{2.8}$$

This monotonic frequency sweep is the basis of the Sobolev sweep algorithm (§3.4.3).

2.5 Bound–Free and Free–Free Opacity

In SN Ia ejecta, continuum opacities are negligible because:

- **Bound–free**: Optical photons (1.5–2.5 eV) are far below the ionization thresholds of the dominant species (Si II: 16.35 eV, Fe II: 16.19 eV, Ca II: 11.87 eV). Only far-UV photons (< 1000 Å) could ionize these ions.
- **Free–free**: Electron densities are very low ($n_e \sim 10^9$ cm^{-3} inner, $\sim 10^6$ cm^{-3} outer), giving $\kappa_{\rm ff} \sim 10^{-20}$ cm^{-1}.

The combined continuum optical depth across a shell is $\tau_{\rm cont} \sim 10^{-6}$ to 10^{-9}, confirming that **line opacity dominates** the spectrum formation in SN Ia.

3 Monte Carlo Photon Transport

3.1 The Indivisible Energy Packet Formalism

LUMINA follows the Lucy [3, 4] formalism, where photons are represented as discrete **energy packets** (r-packets) with properties:

Table 3.1: Properties of an r-packet in LUMINA.

Symbol	Description	Unit
r	Radial position	cm
μ	Direction cosine ($\cos\theta$)	–
ν	Lab-frame frequency	Hz
ε	Packet energy	erg
i_{shell}	Current shell index	–

All packets carry the same energy:

$$\varepsilon_{\text{pkt}} = \frac{L_{\text{inner}}\,\Delta t}{N_{\text{packets}}} \tag{3.1}$$

where L_{inner} is the luminosity at the inner boundary and Δt is the simulation time interval.

3.2 Packet Initialization

Packets are emitted from the photosphere ($r = R_{\text{inner}}$) with:

3.2.1 Frequency Sampling from the Planck Function

The emission frequency is sampled from a blackbody at temperature T_{inner} using the Bjorkman and Wood [1] method:

1. Draw $\xi_0 \sim U(0,1)$
2. Find $l_{\min}$ such that $\sum_{i=1}^{l_{\min}} i^{-4} \geq \frac{\pi^4}{90}\,\xi_0$
3. Draw $\xi_1, \xi_2, \xi_3, \xi_4 \sim U(0,1)$
4. Compute $x = -\ln(\xi_1\xi_2\xi_3\xi_4)/l_{\min}$
5. Set $\nu = x\,k_B T_{\text{inner}}/h$

This is exact (no rejection) and samples directly from $B_\nu(T)$.

3.2.2 Angular Distribution

The direction cosine is sampled from a limb-darkened distribution:

$$\mu = \sqrt{\xi}, \quad \xi \sim U(0,1) \tag{3.2}$$

ensuring that more packets are emitted along the normal direction.

3.3 The Packet Propagation Loop

Each packet undergoes a loop until it escapes ($r > R_{\text{outer}}$) or is reabsorbed ($r < R_{\text{inner}}$ moving inward):

Algorithm: Single Packet Loop

1. **Draw random optical depth**: $\tau_{\text{event}} = -\ln(\xi)$
2. **Compute distances** to possible events:
   - d_{boundary}: distance to next shell wall
   - d_{line}: distance to next Sobolev resonance
   - d_{e}: distance to electron scattering ($\tau = n_e \sigma_T d$)
3. **Select minimum**: $d_{\min} = \min(d_{\text{boundary}}, d_{\text{line}}, d_{\text{e}})$
4. **Move packet**: update r, μ, accumulate estimators
5. **Handle interaction**:
   - Boundary: change shell, check escape/reabsorption
   - Line: scatter, downbranch, or activate macro-atom
   - Electron: Thomson scatter (isotropic re-emission)
6. **Repeat** from step 1

3.4 Distance Calculations

3.4.1 Distance to Shell Boundary

For a packet at (r, μ) in shell $[r_{\text{in}}, r_{\text{out}}]$:

Outward ($\mu > 0$ or $r > r_{\text{in}}/\mu$):

$$d_{\text{out}} = \sqrt{r_{\text{out}}^2 - r^2(1-\mu^2)} - r\mu \tag{3.3}$$

Inward ($\mu < 0$ and impact parameter $< r_{\text{in}}$):

$$d_{\text{in}} = -r\mu - \sqrt{r_{\text{in}}^2 - r^2(1-\mu^2)} \tag{3.4}$$

3.4.2 Distance to Electron Scattering

$$d_e = \frac{\tau_{\text{event}}}{n_e \, \sigma_T} \tag{3.5}$$

3.4.3 Distance to Sobolev Resonance

As the packet traverses the shell, the CMF frequency sweeps through the line list. For a line at rest frequency ν_{line}:

$$d_{\text{line}} = \frac{\nu_{\text{cmf}}(r,\mu) - \nu_{\text{line}}}{\nu_{\text{lab}}} \, c \, t_{\text{exp}} \tag{3.6}$$

Important: The Sobolev sweep must scan from the *entry* frequency to the *exit* frequency of each shell. The original LUMINA code used a fixed ±1% window, which missed lines during thick shell crossings. The corrected algorithm (Task #067) uses the full Doppler sweep, increasing line interactions from 0.6% to > 50% of packet steps.

3.5 Line Interaction Types

When a packet encounters a line, three interaction modes are available:

3.5.1 Resonant Scattering (Mode 0)

The packet is absorbed and re-emitted at the same line frequency with a new random direction:

$$\mu_{\text{new}} \sim U(-1, +1), \quad \nu_{\text{new}} = \nu_{\text{line}} \tag{3.7}$$

3.5.2 Downbranching (Mode 1)

The packet is absorbed into the upper level and re-emitted in a *different* line, selected from the downbranching probability distribution:

$$P(\text{emit in line } j) = \frac{A_{u \to l_j}}{\sum_k A_{u \to l_k}} \tag{3.8}$$

This enables fluorescence: a UV photon absorbed in one line can be re-emitted in the optical.

3.5.3 Macro-Atom (Mode 2)

The full macro-atom formalism [4, 5] activates the atom at the upper energy level and follows a Markov chain of internal transitions:

Algorithm: Macro-Atom Transition Walk

1. Start at activation level l
2. Look up transition block for level l (list of possible transitions)
3. Draw $\xi \sim U(0, 1)$, select transition k from cumulative probabilities
4. If transition type ≥ 0 (internal): move to destination level, go to step 2
5. If transition type < 0 (emission): emit in the associated line, **exit**

Maximum 500 iterations (safety limit).

Emission types include:

- -1: Bound–bound emission (emit photon in a specific line)
- -2: Bound–free emission (photoionization, thermalization)
- -3: Free–free emission (thermal)
- -4: Adiabatic cooling

3.6 Monte Carlo Estimators

As packets propagate, they contribute to estimators that probe the radiation field.

Definition 3.1 — Mean Intensity Estimator.

$$\hat{J}_\nu = \frac{1}{4\pi\, V\, \Delta t} \sum_{\text{packets}} \varepsilon_{\text{cmf}}\, \Delta s \tag{3.9}$$

In practice, two scalar estimators are accumulated per shell:

$$j = \sum_{\text{packets}} \varepsilon_{\text{cmf}}\, \Delta s \tag{3.10}$$

$$\bar{\nu} = \frac{\sum_{\text{packets}} \varepsilon_{\text{cmf}}\, \nu_{\text{cmf}}\, \Delta s}{\sum_{\text{packets}} \varepsilon_{\text{cmf}}\, \Delta s} \tag{3.11}$$

From these, the radiation temperature and dilution factor are recovered:

$$T_{\text{rad}} = T_{\text{rad,const}} \times \frac{\bar{\nu}}{j} \tag{3.12}$$

$$W = \frac{j}{4\,\sigma_{\text{SB}}\, T_{\text{rad}}^4\, \Delta t\, V} \tag{3.13}$$

where $T_{\text{rad,const}} = \frac{\pi^4}{15 \cdot 24 \cdot \zeta(5)} \frac{h}{k_B} = 1.2523 \times 10^{-11}$ K·s.

3.7 Convergence: The Iteration Loop

LUMINA iterates between transport and plasma calculations:

1. **Transport**: Run N_{packets} through the ejecta, accumulating estimators.
2. **Radiation field update**: Compute T_{rad}, W from estimators, with **damping**:

$$X_{\text{new}} = X_{\text{old}} + d \cdot (X_{\text{est}} - X_{\text{old}}), \quad d = 0.5 \tag{3.14}$$

3. **Plasma update**: Recompute ionization, level populations, τ_{Sob}.
4. T_{inner} **update** (after hold iterations):

$$\boxed{T_{\text{inner,new}} = T_{\text{inner,old}} + d \cdot \left[T_{\text{inner}} \left(\frac{L_{\text{emitted}}}{L_{\text{requested}}} \right)^{-0.5} - T_{\text{inner,old}} \right]} \tag{3.15}$$

Remark 3.1 — Exponent -0.5 vs $+0.25$. Naïvely, Stefan–Boltzmann gives $T \propto L^{0.25}$, suggesting the correction factor should be $(L_{\text{em}}/L_{\text{req}})^{0.25}$. However, TARDIS uses the exponent -0.5 because changing T_{inner} also changes the opacity (through ionization), leading to non-linear feedback. The empirical -0.5 accounts for this.

II

Code Architecture

4 Overview & Build System

4.1 File Structure

LUMINA-SN consists of the following source files:

Table 4.1: Source files and their roles.

File	Lines	Purpose
`lumina.h`	378	Master header: all structures, constants, prototypes
`lumina_transport.c`	515	CPU transport kernel
`lumina_plasma.c`	524	Plasma solver & convergence
`lumina_atomic.c`	700	Atomic data loading (HDF5, CSV, NPY)
`lumina_main.c`	466	Main driver & iteration loop
`lumina_cuda.cu`	1353	CUDA GPU transport kernel
Total	**3936**	

4.2 Build System

Listing 4.1: Building LUMINA-SN

```
# CPU build (serial)
make

# CPU build with OpenMP parallelism
make OMP=1

# CUDA GPU build
make cuda
```

Important: `make clean` deletes CSV output files. Always save important spectral outputs before rebuilding.

Table 4.2: Compiler flags.

Target	Compiler	Flags
CPU	`gcc`	`-O2 -Wall -Wextra -std=c11 -lm`
CPU+OMP	`gcc`	adds `-fopenmp`
GPU	`nvcc`	`-O2 -arch=sm_89 -std=c++14`

4.3 Dependencies

- **HDF5** (optional): For loading atomic data from `kurucz_cd23_chianti_H_He.h5`
- **CUDA Toolkit** $\geq$ **12.0**: For GPU builds (tested with CUDA 13.0, sm_89)
- **OpenMP**: For CPU parallelism (optional)
- **Standard C library**: `math.h`, `stdio.h`, `stdlib.h`, `string.h`

4.4 Execution

Listing 4.2: Running LUMINA-SN

```
# 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
```

5 Data Structures

5.1 The `RPacket` Structure

The fundamental unit of the Monte Carlo simulation:

Listing 5.1: `RPacket` — photon energy packet

```c
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;
```

5.2 Geometry

The 1D spherically symmetric ejecta model:

Listing 5.2: `Geometry` — shell structure

```c
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;
```

The radii are derived from velocities via homologous expansion: $r = v \times t_{\text{exp}}$.

5.3 Opacity State

Pre-computed opacity data used during transport:

Listing 5.3: `OpacityState` — line and continuum opacities

```
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;
```

Remark 5.1 — Line List Ordering. Lines are sorted in *descending* frequency order. As a packet's CMF frequency decreases while traversing a shell, it encounters lines from high to low frequency. This ordering enables efficient forward-only scanning.

5.4 Plasma State

Thermodynamic state updated each iteration:

Listing 5.4: `PlasmaState` — radiation field quantities

```
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;
```

5.5 Atomic Data

Comprehensive atomic physics database for the plasma solver:

Listing 5.5: `AtomicData` — atomic physics for Saha–Boltzmann

```
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;
```

5.6 Monte Carlo Estimators

Accumulated during transport, used to update the radiation field:

Listing 5.6: `Estimators` — radiation field accumulators

```
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;
```

Remark 5.2 — GPU Limitation. The `j_blue` and `Edotlu` estimators are *not computed* on the GPU because they require $n_{\text{lines}} \times n_{\text{shells}} \approx 4$ million atomic additions per iteration — prohibitively expensive for `atomicAdd`.

6 Transport Engine

6.1 Overview

The transport engine (`lumina_transport.c`, 515 lines) propagates r-packets through the ejecta. It is the most performance-critical component.

6.2 The `trace_packet` Function

This function computes the next interaction event for a packet:

1. Compute distance to shell boundaries (d_{boundary})
2. Scan the Sobolev line list for resonances (d_{line}, accumulated τ)
3. Compute distance to electron scattering ($d_e = \tau_{\text{event}}/(n_e \sigma_T)$)
4. Return the minimum distance and interaction type

6.3 Sobolev Line Sweep

The sweep algorithm processes lines in descending frequency order:

Listing 6.1: Sobolev sweep (simplified)

```
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;
    }
}
```

6.4 Thomson Scattering

Elastic scattering with free electrons:

1. Transform packet energy/frequency to comoving frame at current angle
2. Sample new direction: $\mu_{\text{new}} \sim U(-1, +1)$ (isotropic in CMF)
3. Transform back to lab frame with new angle
4. Energy is conserved in the comoving frame

$$\varepsilon_{\text{lab,new}} = \varepsilon_{\text{cmf}} \times \frac{1}{1 - \mu_{\text{new}} \cdot v/c} \tag{6.1}$$

6.5 Boundary Crossing

When a packet crosses a shell boundary:

1. Update shell index: $i_{\text{shell}} \leftarrow i_{\text{shell}} + \delta$ where $\delta = +1$ (outward) or -1 (inward)
2. Nudge position by $\epsilon = 10^{-10} \times \Delta r_{\text{shell}}$ into the new shell
3. Check for escape ($i > n_{\text{shells}} - 1$) or reabsorption ($i < 0$, inward-moving)

Important: The position nudge is critical. Without it, the packet lands exactly on the boundary, and the next distance calculation returns $d = 0$, causing an infinite loop (Task #024).

7 Plasma Physics Solver

7.1 Overview

The plasma solver (`lumina_plasma.c`, 524 lines) computes the thermodynamic state of the ejecta for each iteration. It implements the TARDIS-compatible nebular approximation [6].

7.2 Step 1: Partition Functions

The partition function for ion (Z, stage) in shell s is:

$$\mathcal{Z}(Z, \text{stage}, s) = \underbrace{\sum_{i \in \text{meta}} g_i \, e^{-E_i / k_B T_{\text{rad}}(s)}}_{\mathcal{Z}_{\text{meta}}} + W(s) \cdot \underbrace{\sum_{i \in \text{non-meta}} g_i \, e^{-E_i / k_B T_{\text{rad}}(s)}}_{\mathcal{Z}_{\text{non}}} \tag{7.1}$$

Important: The Boltzmann factors use T_{rad} for *all* levels (both metastable and non-metastable). Earlier code incorrectly used T_e for metastable levels. The dilution factor W suppresses the non-metastable contribution at large distances from the photosphere.

7.3 Step 2: Electron Density

Computed iteratively with TARDIS-style damping:

Algorithm: Electron Density Iteration

1. Start with initial guess $n_e^{(0)}$
2. For each element: compute ionization ratios using nebular Saha (Eq. 7.2)
3. Normalize ion populations to element abundance
4. Compute $n_e^{(\text{calc})} = \sum_{\text{ions}} \text{stage} \times n_{\text{ion}}$
5. Damped update: $n_e^{(k+1)} = 0.5 \times n_e^{(\text{calc})} + 0.5 \times n_e^{(k)}$
6. Convergence: $|n_e^{(k+1)} - n_e^{(k)}| / n_e^{(k)} < 0.05$

7.4 Step 3: Nebular Saha Ionization

The ionization ratio between consecutive stages is:

$$\boxed{\frac{n_{i+1}}{n_i} = \frac{\Phi_{\text{neb}}}{n_e}} \tag{7.2}$$

where the nebular ionization coefficient Φ_{neb} is:

$$\Phi_{\text{neb}} = \Phi_{\text{LTE}} \times W \times [\zeta \cdot \delta + W \cdot (1 - \zeta)] \times \sqrt{\frac{T_e}{T_{\text{rad}}}} \tag{7.3}$$

$$\Phi_{\text{LTE}} = \frac{\mathcal{Z}_{i+1}}{\mathcal{Z}_i} \times 2 \times g_e \times e^{-\chi/k_B T_{\text{rad}}} \tag{7.4}$$

$$g_e = \left(\frac{2\pi m_e k_B T_{\text{rad}}}{h^2}\right)^{3/2} \tag{7.5}$$

$$\delta = \frac{T_e}{T_{\text{rad}}} \exp\left[\chi\left(\frac{1}{k_B T_{\text{rad}}} - \frac{1}{k_B T_e}\right)\right] \tag{7.6}$$

Here χ is the ionization energy, and ζ is a non-LTE correction factor interpolated from tabulated values.

7.5 Step 4: τ_{Sobolev} Update

With ion populations known, τ_{Sob} is recomputed for each line and shell using Eq. (2.2). The level populations follow the Boltzmann distribution within each ion:

$$n_{l,\text{meta}} = \frac{g_l}{\mathcal{Z}} n_{\text{ion}} e^{-E_l/k_B T_{\text{rad}}}, \quad n_{l,\text{non}} = W \times \frac{g_l}{\mathcal{Z}} n_{\text{ion}} e^{-E_l/k_B T_{\text{rad}}} \tag{7.7}$$

7.6 Step 5: Radiation Field Update

From the MC estimators j and $\overline{\nu}$ (Eqs. 3.10–3.11):

$$T_{\text{rad,est}}(s) = 1.2523 \times 10^{-11} \times \frac{\overline{\nu}(s)}{j(s)} \quad [\text{K}] \tag{7.8}$$

$$W_{\text{est}}(s) = \frac{j(s)}{4\,\sigma_{\text{SB}}\,T_{\text{rad}}^4(s)\,\Delta t\,V(s)} \tag{7.9}$$

All quantities are damped:

$$X_{\text{new}} = X_{\text{old}} + 0.5 \times (X_{\text{est}} - X_{\text{old}}) \tag{7.10}$$

8 Atomic Data System

8.1 Data Sources

LUMINA uses the TARDIS reference atomic dataset, originally from Kurucz CD23 and CHIANTI:

Table 8.1: Atomic data files.

File	Format	Contents
`line_list.csv`	CSV	ν, Z, ion, f_{lu}, λ per line
`levels.csv`	CSV	E (eV), g, metastable flag per level
`ionization_energies.csv`	CSV	χ per ion
`tau_sobolev.npy`	NPY	τ_{Sob} reference values
`transition_probabilities.npy`	NPY	Macro-atom transition probs
`macro_atom_data.csv`	CSV	Transition types, destinations
`zeta_data.npy`	NPY	Non-LTE correction factors
`abundances.csv`	CSV	Mass fractions per shell

8.2 The NPY Format Reader

LUMINA includes a custom NPY reader (no NumPy dependency in C):

1. Read 6-byte magic: `\x93NUMPY`
2. Read version (1 or 2) and header length
3. Parse Python dict header for `shape`, `dtype`, `fortran_order`
4. Read raw binary data
5. Transpose if Fortran-ordered

8.3 The CSV Parser

Important: The `macro_atom_data.csv` header starts with an unnamed index column: "`,atomic_number,...`". The standard `strtok()` function skips leading delim-

iters, causing a column offset of -1. LUMINA uses a manual field-by-field parser that handles empty fields correctly.

9 CUDA GPU Implementation

9.1 Design Philosophy

The GPU implementation maps **one CUDA thread per packet**. Each thread independently propagates its packet through the ejecta, requiring no inter-thread communication except for atomic estimator updates.

9.2 Memory Layout

Table 9.1: GPU memory allocation.

Data	Access	Size (200K pkts)
Line frequencies	Read-only	$n_{\text{lines}} \times 8$ B
τ_{Sob}	Read-only	$n_{\text{lines}} \times n_{\text{shells}} \times 8$ B
Transition probs	Read-only	$n_{\text{trans}} \times n_{\text{shells}} \times 8$ B
Shell geometry	Read-only	$n_{\text{shells}} \times 4 \times 8$ B
RNG states	Read/write	$N_{\text{pkt}} \times 4 \times 8$ B
j, $\bar{\nu}$ estimators	Atomic write	$n_{\text{shells}} \times 2 \times 8$ B
Output arrays	Write-only	$N_{\text{pkt}} \times 3 \times 8$ B

Total GPU memory: approximately 2 GB for a typical run.

9.3 Kernel Launch Configuration

Listing 9.1: Kernel launch

```
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>>>(...);
```

Important: The original code had `CUDA_MAX_BLOCKS = 1024`, limiting execution to 262K packets regardless of N_{packets}. For 2M packets, only 13% executed, causing $j_{\text{estimator}}$ to be 76× too low. This was the single largest GPU bug (Task #13).

9.4 Random Number Generation

Each thread uses an independent **xoshiro256**** generator with 256 bits of state (4 × `uint64`). Seeds are derived from the packet index via SplitMix64:

Listing 9.2: Per-thread RNG initialization

```
__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);
}
```

9.5 Atomic Estimator Updates

The j and $\bar{\nu}$ estimators are updated using CUDA `atomicAdd`:

Listing 9.3: Estimator accumulation on GPU

```
__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);
}
```

Since there are only $n_{\text{shells}} = 30$ accumulation targets, contention is manageable.

9.6 Performance

Table 9.2: CPU vs GPU performance (NVIDIA RTX 5000 Ada, sm_89).

N_{packets}	**CPU (1 core)**	**CPU (OMP64)**	**GPU**	**Speedup**
20,000	0.7 s	0.1 s	0.08 s	9×
200,000	7.2 s	1.1 s	0.73 s	10×
2,000,000	72 s	11 s	7.3 s	10×
20,000,000	720 s	110 s	73 s	10×

Remark 9.1 — Statistical Accuracy. GPU and CPU produce statistically identical results. At 200K packets: W error 1.06%, T_{rad} error 0.58% (relative to TARDIS reference). The scaling follows $\sigma \propto N^{-0.35}$ to $N^{-0.40}$, close to Poisson ($N^{-0.5}$).

9.7 Resolved GPU Bugs

Four critical bugs were identified and fixed during development:

1. **MAX_BLOCKS = 1024**: Only 262K threads could launch. Fixed to 131072.
2. **Shared memory race**: `__shared__ ShellCache` shared by all 256 threads, but only thread 0 loaded data. Fixed: use L1-cached global memory.
3. **Counter accumulation**: Escape/reabsorb counters not reset between iterations. Fixed: explicit reset in `cuda_reset_estimators()`.
4. **Boundary sticking**: Packets land exactly on shell boundaries due to floating-point precision. Fixed: explicit nudge by $10^{-10} \times \Delta r$.

Usage & Applications

10 Installation & Quick Start

10.1 Prerequisites

Table 10.1: System requirements.

Component	Requirement
C Compiler	GCC $\geq$ 9.0 (C11 support)
CUDA (optional)	Toolkit $\geq$ 12.0, compute capability $\geq$ 7.0
HDF5 (optional)	`libhdf5-dev` for atomic data loading
Memory	$\geq$ 4 GB RAM (CPU), $\geq$ 4 GB VRAM (GPU)

10.2 Step-by-Step Setup

Listing 10.1: Complete setup procedure

```
# 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
```

10.3 Input Directory Structure

LUMINA expects a reference data directory (default: `data/tardis_reference/`) containing:

Listing 10.2: Required input files

```
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
```

10.4 Output Files

Table 10.2: Output files produced by LUMINA.

File	Contents
`lumina_spectrum.csv`	λ (Å) vs L_λ (erg/s/cm)
`lumina_plasma_state.csv`	Final W, $T_{\rm rad}$ per shell
`lumina_tau_validation.csv`	$\tau_{\rm Sob}$ at shell 0 (debug)

11 The Ejecta Model

11.1 Three-Zone Composition

LUMINA uses a three-zone abundance structure motivated by SN Ia nucleosynthesis:

Table 11.1: Default three-zone composition model.

Zone	Fe	Si	S	Ca	Co	Ni	C	O
Core ($v < v_{\rm core}$)	*free*	0.05	0.05	0.03	0.05	*free*	0.02	filler
Wall ($v_{\rm core}$–$v_{\rm wall}$)	*free*	*free*	0.05	0.03	0.05	*free*	0.02	filler
Outer ($v > v_{\rm wall}$)	*free*	0.02	0.02	0.01	0.05	*free*	0.02	filler

"Filler" means oxygen fills the remaining mass fraction to ensure $\sum X_i = 1$.

Important: Oxygen is the correct filler element for the outer zone because it has very few optical absorption lines and is essentially transparent. Using Fe/Ni/S as fillers creates massive line blanketing ($> 10^6$ active lines) that produces an artificial pseudo-photosphere (Task #063).

11.2 Broken Power-Law Density

The density profile is a broken power law:

$$\rho(v) = \begin{cases} \rho_0 \left(\dfrac{v}{v_{\rm inner}} \right)^{n_{\rm inner}} & v < v_{\rm break} \\ \rho_{\rm break} \left(\dfrac{v}{v_{\rm break}} \right)^{n_{\rm outer}} & v \geq v_{\rm break} \end{cases} \tag{11.1}$$

where $\rho_{\rm break} = \rho_0 (v_{\rm break}/v_{\rm inner})^{n_{\rm inner}}$ ensures continuity at $v_{\rm break}$.

Typical values: $n_{\rm inner} \approx -7$, $n_{\rm outer} \approx -10$.

11.3 Physical Parameter Space

LUMINA-ML (the machine learning emulator companion) uses a 15-dimensional parameter space:

Table 11.2: Full 15D parameter space with ranges.

#	Parameter	Min	Max	Description
1	$\log L$	42.50	43.50	Luminosity [erg/s]
2	$v_{\rm inner}$	7000	15000	Photosphere velocity [km/s]
3	$\log \rho_0$	-14.0	-12.3	Reference density [g/cm^3]
4	$n_{\rm inner}$	-10	-4	Inner density exponent
5	$T_e/T_{\rm rad}$	0.7	1.0	Temperature ratio
6	$v_{\rm core}$	9000	17000	Core/wall boundary [km/s]
7	$v_{\rm wall}$	12000	24000	Wall/outer boundary [km/s]
8	$X_{\rm Fe,core}$	0.05	0.85	Core iron abundance
9	$X_{\rm Si,wall}$	0.05	0.75	Wall silicon abundance
10	$v_{\rm break}$	10000	22000	Density break velocity [km/s]
11	$n_{\rm outer}$	-14	-4	Outer density exponent
12	$t_{\rm exp}$	10	28	Time since explosion [days]
13	$X_{\rm Fe,wall}$	0.001	0.50	Wall iron contamination
14	$X_{\rm Ni}$	0.005	0.25	Nickel abundance (all zones)
15	$X_{\rm Fe,outer}$	0.001	0.15	Outer iron abundance

12 Parameter Fitting

12.1 Fitting Strategy

LUMINA includes a multi-phase parameter search framework (`scripts/fit_parameter_search.py`) that combines Latin Hypercube Sampling with progressive refinement.

12.1.1 Phase 1: Coarse Exploration

- 200 Latin Hypercube samples across full parameter space
- 20K packets × 5 iterations (fast, ∼5 s per model)
- Score by feature-weighted RMS
- Select top-20 candidates

12.1.2 Phase 2: Refinement

- Top-20 candidates re-simulated with 100K packets × 10 iterations
- Better statistics reduce Monte Carlo noise
- Select top-3

12.1.3 Phase 3: Production

- Top-3 candidates with 500K packets × 20 iterations
- Highest-fidelity spectra
- Final selection based on composite score

12.2 Objective Function

The composite scoring function includes:

$$\text{Score} = \text{RMS}_{\text{spec}} + 0.5\,|\Delta d_{\text{Si II}}| + 0.2\,|\Delta \log v_{\text{Si II}}| + 0.1\,|\Delta \lambda_{\text{min}}| \tag{12.1}$$

where:

- RMS_{spec}: Spectral RMS over 5000–8000 Å
- $\Delta d_{\text{Si II}}$: Si II 6355 absorption depth error
- $\Delta \log v_{\text{Si II}}$: Si II velocity error (log scale)
- $\Delta \lambda_{\text{min}}$: Si II trough wavelength error

13 Physical Constants & Reference Values

Table 13.1: Physical constants used in LUMINA (CGS).

Symbol	Description	Value
c	Speed of light	$2.99792458 \times 10^{10}$ cm/s
h	Planck constant	$6.62607015 \times 10^{-27}$ erg·s
k_B	Boltzmann constant	1.380649×10^{-16} erg/K
σ_{SB}	Stefan–Boltzmann	5.670374×10^{-5} erg/cm^2/s/K^4
σ_T	Thomson cross-section	6.6525×10^{-25} cm^2
m_e	Electron mass	9.10938×10^{-28} g
e	Electron charge	4.80321×10^{-10} esu
$\pi e^2/(m_e c)$	Sobolev coefficient	2.6540×10^{-2} cm^2/s
$T_{\mathrm{rad,const}}$	T_{rad} estimator constant	1.2523×10^{-11} K·s

Bibliography

[1] J. E. Bjorkman and K. Wood. "Radiative Equilibrium and Temperature Correction in Monte Carlo Radiation Transfer". In: *The Astrophysical Journal* 554 (2001), pp. 615–623.

[2] W. E. Kerzendorf and S. A. Sim. "A spectral synthesis code for rapid modelling of supernovae". In: *Monthly Notices of the Royal Astronomical Society* 440 (2014), pp. 387–404. DOI: 10.1093/mnras/stu055.

[3] L. B. Lucy. "Improved Monte Carlo techniques for the spectral synthesis of supernovae". In: *Astronomy & Astrophysics* 345 (1999), pp. 211–220.

[4] L. B. Lucy. "Monte Carlo techniques for time-dependent radiative transfer in 3-D supernovae". In: *Astronomy & Astrophysics* 384 (2002), pp. 725–735.

[5] L. B. Lucy. "Monte Carlo transition probabilities". In: *Astronomy & Astrophysics* 403 (2003), pp. 261–275.

[6] P. A. Mazzali and L. B. Lucy. "The application of Monte Carlo methods to the synthesis of early-time supernovae spectra". In: *Astronomy & Astrophysics* 279 (1993), pp. 447–456.

[7] K. Nomoto, F.-K. Thielemann, and K. Yokoi. "Accreting white dwarf models of Type I supernovae. III. Carbon deflagration supernovae". In: *The Astrophysical Journal* 286 (1984), pp. 644–658.

[8] M. M. Phillips. "The absolute magnitudes of Type IA supernovae". In: *The Astrophysical Journal Letters* 413 (1993), pp. L105–L108.

Index