```
==========================================================
#            Hilios v18 - Dark Sector Physics
# ==========================================================

# ============================================================
# ===          PART 1: IMPORTS & PARAMETERS          ===
# ============================================================
import numpy as np
import cupy as cp
import time
import h5py
import os

import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from IPython.display import display, clear_output

sim_params = {
    # Simulation Control
    "NUM_STEPS": 50000, "VIS_INTERVAL": 100, "DT": 1e-5,
    "GRID_SIZE": 101,

    # RDU Physics Constants
    "G_RDU": 0.05, "K_S": 1.0,
    "C_ACCRETION_DRAG": 0.1, "ACCRETION_RADIUS": 15.0, "C_DYN_FRICTION": 0.01,

    # Knoechelman Mechanism Constants
    "A_CONST": 1.2, "N_SENSITIVITY": 0.5, "C_UNIFY": 1.5e-5,
    "K_ROT": 3.0, "C_BALANCE": 423.0, "TACHYON_MASS": 0.001,

    # Supernova & Magnetar Parameters
    "SUPERNOVA_SHELL_PARTICLES": 200, "SUPERNOVA_EJECTION_VELOCITY": 0.5,
    "REMNANT_MASS_FRACTION": 0.2, "MAGNETAR_STRENGTH": 500.0,
    "GAS_CHARGE": 0.05,

    # Dark Strong Force Parameters
    "DARK_FORCE_STRENGTH": 0.01,  # Epsilon_DS: The overall strength
    "REPULSION_RADIUS": 2.5,      # r_0: The distance where the force turns repulsive

    # Particle Population
    "gas_particle_count": 15000,
    "star_population": {"G-Star": 4000, "BD": 3000, "WD": 2500, "NS": 500, "Magnetar": 0},
    "NUM_TACHYONS": 5000,
```

```
    # Output Control
    "OUTPUT_FILENAME": "hilios_v18_final_state.h5",
    "OUTPUT_DIR": "/content/drive/MyDrive/RDU_Sims/"
}


# ============================================================
# ===              PART 2: CUDA KERNELS              ===
# ============================================================
merger_kernel = cp.RawKernel(r'''
extern "C" __global__
void merger_kernel(float* pos, float* vel, float* mass, float* radius, int* is_consumed, int
num_particles,
                  int* merger_log, int* merger_log_count, int max_log_size) {
    int i = blockDim.x * blockIdx.x + threadIdx.x; if (i >= num_particles || is_consumed[i] == 1)
return;
    for (int j = i + 1; j < num_particles; ++j) {
        if (is_consumed[j] == 1) continue;
        float dx = pos[i*3+0] - pos[j*3+0]; float dy = pos[i*3+1] - pos[j*3+1]; float dz = pos[i*3+2] -
pos[j*3+2];
        float dist_sq = dx*dx + dy*dy + dz*dz; float combined_radius = radius[i] + radius[j];
        if (dist_sq < combined_radius * combined_radius) {
            int survivor_idx = (mass[i] >= mass[j]) ? i : j; int consumed_idx = (mass[i] >= mass[j]) ? j :
i;
            if (atomicCAS(&is_consumed[consumed_idx], 0, 1) == 0) {
                float m_survivor = mass[survivor_idx]; float m_consumed = mass[consumed_idx];
float total_mass = m_survivor + m_consumed;
                vel[survivor_idx*3+0] = (m_survivor * vel[survivor_idx*3+0] + m_consumed *
vel[consumed_idx*3+0]) / total_mass;
                vel[survivor_idx*3+1] = (m_survivor * vel[survivor_idx*3+1] + m_consumed *
vel[consumed_idx*3+1]) / total_mass;
                vel[survivor_idx*3+2] = (m_survivor * vel[survivor_idx*3+2] + m_consumed *
vel[consumed_idx*3+2]) / total_mass;
                atomicAdd(&mass[survivor_idx], m_consumed);
                radius[survivor_idx] = cbrtf(powf(radius[survivor_idx], 3) + powf(radius[consumed_idx],
3));
                int log_idx = atomicAdd(merger_log_count, 1);
                if (log_idx < max_log_size) { merger_log[log_idx * 2 + 0] = survivor_idx;
merger_log[log_idx * 2 + 1] = consumed_idx; }
            }
        }
    }
}
''', 'merger_kernel')
```

```
stellar_evolution_kernel = cp.RawKernel(r'''
extern "C" __global__
void stellar_evolution_kernel(int* particle_types, float* lifetimes, int* event_flags, int n, float dt) {
    int i = blockDim.x * blockIdx.x + threadIdx.x; if (i >= n || lifetimes[i] <= 0) return; lifetimes[i] -=
dt;
    if (lifetimes[i] <= 0) { if (particle_types[i] == 2) { particle_types[i] = 4; event_flags[i] = 1; } }
}
''', 'stellar_evolution_kernel')
knoechelman_kernel = cp.RawKernel(r'''
extern "C" __global__
void knoechelman_kernel(const float* positions, float* velocities, float* masses, const float*
radii, const float* spins,
    int* particle_types, int* is_consumed, int* event_flags, int n, float A, float N_sens, float
C_unify,
    float k_rot, float C_balance, float G_rdu, float tachyon_mass) {
    int i = blockDim.x * blockIdx.x + threadIdx.x; if (i >= n || is_consumed[i] == 1 || particle_types[i]
== 0 || particle_types[i] >= 10) return;
    float drc_pos_x = positions[0], drc_pos_y = positions[1], drc_pos_z = positions[2]; float
drc_radius = radii[0];
    float dx = positions[i*3+0] - drc_pos_x; float dy = positions[i*3+1] - drc_pos_y; float dz =
positions[i*3+2] - drc_pos_z;
    float dist_sq = dx*dx + dy*dy + dz*dz;
    if (dist_sq < (radii[i] + drc_radius) * (radii[i] + drc_radius)) {
        if (atomicExch(&is_consumed[i], 1) == 0) {
            float drc_mass = masses[0]; float drc_spin_mag = sqrtf(spins[0]*spins[0] +
spins[1]*spins[1] + spins[2]*spins[2]);
            float I_over_Imax_proxy = fminf(1.0f, drc_spin_mag / 100.0f);
            float drc_density = (drc_radius > 0) ? drc_mass / (4.1887f * drc_radius * drc_radius *
drc_radius) : 0.0f;
            float phi_U = C_unify * (drc_mass * drc_density) * expf(k_rot * I_over_Imax_proxy);
            float consumed_mass = masses[i]; float omega_R = A * powf(phi_U /
fmaxf(consumed_mass, 1e-9f), N_sens);
            float e_trap = (drc_radius > 0) ? G_rdu * (drc_mass * drc_mass) / drc_radius : 1e9f;
            float xi = (e_trap > 0) ? C_balance * (omega_R * omega_R) / e_trap : 0.0f;
            if (xi > 1.0f) { particle_types[i] = 11; velocities[i*3+0]*=50.0f; velocities[i*3+1]*=50.0f;
velocities[i*3+2]*=50.0f; }
            else { particle_types[i] = 10; velocities[i*3+0]*=5.0f; velocities[i*3+1]*=5.0f;
velocities[i*3+2]*=5.0f; }
            masses[i] = tachyon_mass; event_flags[i] = 2;
        }
    }
}
''', 'knoechelman_kernel')
```

```
deposit_mass_kernel = cp.RawKernel(r'''extern "C" __global__ void deposit_mass_kernel(const
float* positions, const float* masses, float* grid, int n, int gs) { int
i=blockDim.x*blockIdx.x+threadIdx.x; if(i>=n) return; int x=(int)positions[i*3+0],
y=(int)positions[i*3+1], z=(int)positions[i*3+2]; if(x>=0&&x<gs&&y>=0&&y<gs&&z>=0&&z<gs) {
atomicAdd(&grid[x*gs*gs+y*gs+z], masses[i]); } }''', 'deposit_mass_kernel')
update_chronos_kernel = cp.RawKernel(r'''extern "C" __global__ void
update_chronos_kernel(float* C_n_plus_1, const float* C_n, const float* C_n_minus_1, const
float* source_term, int gs, float dt, float K_S, float G) { int x=blockIdx.x*blockIdx.x+threadIdx.x,
y=blockIdx.y*blockDim.y+threadIdx.y, z=blockIdx.z*blockDim.z+threadIdx.z;
if(x>=gs||y>=gs||z>=gs) return; if(x>0&&x<gs-1&&y>0&&y<gs-1&&z>0&&z<gs-1) { int
idx=x*gs*gs+y*gs+z; float
laplacian=K_S*(C_n[idx+gs*gs]+C_n[idx-gs*gs]+C_n[idx+gs]+C_n[idx-gs]+C_n[idx+1]+C_n[idx-
1]-6.0f*C_n[idx]); float source=-G*source_term[idx];
C_n_plus_1[idx]=2.0f*C_n[idx]-C_n_minus_1[idx]+dt*dt*(laplacian+source); } }''',
'update_chronos_kernel')
generate_b_field_kernel = cp.RawKernel(r'''extern "C" __global__ void
generate_b_field_kernel(float* b_field_grid, const float* positions, const float* spins, const int*
particle_types, int n, int gs, float b_strength) { int x=blockIdx.x*blockIdx.x+threadIdx.x,
y=blockIdx.y*blockDim.y+threadIdx.y, z=blockIdx.z*blockDim.z+threadIdx.z;
if(x>=gs||y>=gs||z>=gs) return; int grid_idx=(x*gs*gs+y*gs+z)*3; float bx=0.0f, by=0.0f, bz=0.0f;
for(int i=0; i<n; ++i){if(particle_types[i]==7){ float
mx=positions[i*3+0],my=positions[i*3+1],mz=positions[i*3+2]; float
rx=(float)x-mx,ry=(float)y-my,rz=(float)z-mz; float dist_sq=rx*rx+ry*ry+rz*rz;
if(dist_sq>1e-6f){float strength=b_strength/(dist_sq*sqrtf(dist_sq)); float
sx=spins[i*3+0],sy=spins[i*3+1],sz=spins[i*3+2]; float smag=sqrtf(sx*sx+sy*sy+sz*smag);
if(smag>1e-6f){bx+=strength*sx/smag;by+=strength*sy/smag;bz+=strength*sz/smag;}}}}
b_field_grid[grid_idx+0]=bx; b_field_grid[grid_idx+1]=by; b_field_grid[grid_idx+2]=bz; }''',
'generate_b_field_kernel')
force_kernel = cp.RawKernel(r'''
extern "C" __global__
void force_kernel(
    float* accelerations, const float* positions, const float* velocities, const float* masses,
    const int* particle_types, const float* chronos_field, const float* b_field_grid,
    int n, int gs, float C_drag, float R_drag, float C_friction, float q,
    float dsf_strength, float dsf_radius) {

    int i = blockIdx.x * blockDim.x + threadIdx.x; if (i >= n) return;
    float px=positions[i*3+0], py=positions[i*3+1], pz=positions[i*3+2];
    float ax=0.0f, ay=0.0f, az=0.0f;
    int x=(int)px, y=(int)py, z=(int)pz;
    if (x>0&&x<gs-1&&y>0&&y<gs-1&&z>0&&z<gs-1) {
        ax=-(chronos_field[(x+1)*gs*gs+y*gs+z]-chronos_field[(x-1)*gs*gs+y*gs+z])/2.0f;
        ay=-(chronos_field[x*gs*gs+(y+1)*gs+z]-chronos_field[x*gs*gs+(y-1)*gs+z])/2.0f;
        az=-(chronos_field[x*gs*gs+y*gs+(z+1)]-chronos_field[x*gs*gs+y*gs+(z-1)])/2.0f;
```

```
    }
    int p_type = particle_types[i];
    if (p_type == 1) {
        if (x>=0&&x<gs&&y>=0&&y<gs&&z>=0&&z<gs) {
            int b_idx=(x*gs*gs+y*gs+z)*3; float bx=b_field_grid[b_idx+0], by=b_field_grid[b_idx+1],
bz=b_field_grid[b_idx+2];
            float vx=velocities[i*3+0], vy=velocities[i*3+1], vz=velocities[i*3+2];
            ax+=q*(vy*bz-vz*by); ay+=q*(vz*bx-vx*bz); az+=q*(vx*by-vy*bx);
        }
        float d_sq_c=(50.5f-px)*(50.5f-px)+(50.5f-py)*(50.5f-py)+(50.5f-pz)*(50.5f-pz);
        if(d_sq_c <
R_drag*R_drag){ax-=C_drag*velocities[i*3+0];ay-=C_drag*velocities[i*3+1];az-=C_drag*velociti
es[i*3+2];}
    }
    if (masses[i] > 10.0) {
        float
v_sq=velocities[i*3+0]*velocities[i*3+0]+velocities[i*3+1]*velocities[i*3+1]+velocities[i*3+2]*veloci
ties[i*3+2];
        if(v_sq>0.1f){float f_mag=C_friction*masses[i]/v_sq;
ax-=f_mag*velocities[i*3+0];ay-=f_mag*velocities[i*3+1];az-=f_mag*velocities[i*3+2];}
    }
    if (p_type >= 10) {
        for (int j=0; j<n; ++j) {
            if (i==j || particle_types[j]<10) continue;
            float dx=px-positions[j*3+0], dy=py-positions[j*3+1], dz=pz-positions[j*3+2];
            float r_sq=dx*dx+dy*dy+dz*dz;
            if (r_sq > 1e-6f && r_sq < 25.0f) {
                float r=sqrtf(r_sq); float r0_over_r_6=powf(dsf_radius/r,6.0f); float
r0_over_r_12=r0_over_r_6*r0_over_r_6;
                float force_mag=(12.0f*dsf_strength/r)*(r0_over_r_12-r0_over_r_6);
                ax+=force_mag*(dx/r)/masses[i]; ay+=force_mag*(dy/r)/masses[i];
az+=force_mag*(dz/r)/masses[i];
            }
        }
    }
    accelerations[i*3+0]=ax; accelerations[i*3+1]=ay; accelerations[i*3+2]=az;
}
''', 'force_kernel')

# ============================================================
# === PART 3: DATA INITIALIZATION & GPU TRANSFER      ===
# ==========================================================
print("Initializing Hilios v18..."); star_counts=sim_params["star_population"];
num_stars=sum(star_counts.values())
```

```
num_baryons=sim_params["gas_particle_count"]+num_stars+1;
num_tachyons=sim_params["NUM_TACHYONS"]
total_particles=num_baryons+num_tachyons; gs=sim_params["GRID_SIZE"]
positions_cpu=np.zeros((total_particles,3),dtype=np.float32);
velocities_cpu=np.zeros((total_particles,3),dtype=np.float32)
masses_cpu=np.ones(total_particles,dtype=np.float32);
radii_cpu=np.ones(total_particles,dtype=np.float32)*0.5
is_consumed_cpu=np.zeros(total_particles,dtype=np.int32);
spin_cpu=np.zeros((total_particles,3),dtype=np.float32)
particle_type_cpu=np.zeros(total_particles,dtype=np.int32);
lifetimes_cpu=np.zeros(total_particles,dtype=np.float32)
print(f"Creating {total_particles} particles..."); particle_type_cpu[0]=0; masses_cpu[0]=50000.0;
radii_cpu[0]=2.0
spin_cpu[0]=[0,0,100.0]; positions_cpu[0]=[gs/2,gs/2,gs/2]; disk_indices=slice(1,num_baryons)
num_disk_particles=num_baryons-1;
radius=np.random.uniform(gs*0.1,gs*0.6,num_disk_particles)
angle=np.random.uniform(0,2*np.pi,num_disk_particles);
z_pos=np.random.uniform(-gs*0.02,gs*0.02,num_disk_particles)
positions_cpu[disk_indices,0]=gs/2+radius*np.cos(angle);
positions_cpu[disk_indices,1]=gs/2+radius*np.sin(angle)
positions_cpu[disk_indices,2]=gs/2+z_pos;
orbital_v=np.sqrt(sim_params["G_RDU"]*masses_cpu[0]/radius)
velocities_cpu[disk_indices,0]=-orbital_v*np.sin(angle);
velocities_cpu[disk_indices,1]=orbital_v*np.cos(angle)
start_idx=1; particle_type_cpu[start_idx:start_idx+sim_params["gas_particle_count"]]=1;
start_idx+=sim_params["gas_particle_count"]
type_map={"G-Star":2,"BD":3,"WD":4,"NS":5,"Magnetar":7}; label_map={v:k for k,v in
type_map.items()}
label_map[0]="DRC"; label_map[1]="Gas"
for star_type, count in star_counts.items():
   end_idx=start_idx+count; particle_type_cpu[start_idx:end_idx]=type_map[star_type]
   if star_type=="NS":
lifetimes_cpu[start_idx:end_idx]=np.random.uniform(0.1,sim_params["NUM_STEPS"]*sim_para
ms["DT"])
   start_idx=end_idx
tachyon_indices=slice(num_baryons,total_particles);
positions_cpu[tachyon_indices]=np.random.rand(num_tachyons,3)*gs
velocities_cpu[tachyon_indices]=(np.random.rand(num_tachyons,3)-0.5)*100;
particle_type_cpu[tachyon_indices]=10
print("Transferring data to GPU..."); positions_gpu=cp.asarray(positions_cpu);
velocities_gpu=cp.asarray(velocities_cpu)
masses_gpu=cp.asarray(masses_cpu); radii_gpu=cp.asarray(radii_cpu);
is_consumed_gpu=cp.asarray(is_consumed_cpu)
```

```
spin_gpu=cp.asarray(spin_cpu); particle_type_gpu=cp.asarray(particle_type_cpu);
lifetimes_gpu=cp.asarray(lifetimes_cpu)
accelerations_gpu=cp.empty_like(positions_gpu);
event_flags_gpu=cp.zeros(total_particles,dtype=cp.int32)
chronos_field_n_minus_1=cp.zeros((gs,gs,gs),dtype=cp.float32);
chronos_field_n=cp.zeros((gs,gs,gs),dtype=cp.float32)
chronos_field_n_plus_1=cp.zeros((gs,gs,gs),dtype=cp.float32);
source_term_gpu=cp.zeros((gs,gs,gs),dtype=cp.float32)
b_field_grid=cp.zeros((gs,gs,gs,3),dtype=cp.float32)
MAX_MERGERS_PER_STEP=100;
merger_log_gpu=cp.zeros((MAX_MERGERS_PER_STEP,2),dtype=cp.int32);
merger_log_count_gpu=cp.zeros(1,dtype=cp.int32)
print("Initialization complete.")

# ==========================================================
# ===        PART 4: LIVE SCIENCE DASHBOARD         ===
# ==========================================================
fig=plt.figure(figsize=(16,12)); fig.patch.set_facecolor('#0d0d0d');
dashboard_gs=gridspec.GridSpec(2,4,height_ratios=[3,1.2])
ax_main=fig.add_subplot(dashboard_gs[0,:],projection='3d');
ax_hist_baryon=fig.add_subplot(dashboard_gs[1,0])
ax_hist_tachyon=fig.add_subplot(dashboard_gs[1,1]);
ax_angular_momentum=fig.add_subplot(dashboard_gs[1,2])
ax_log=fig.add_subplot(dashboard_gs[1,3]); plt.subplots_adjust(wspace=0.3,hspace=0.5)
def update_dashboard(step, positions, velocities, particle_types, ang_mom_history,
event_log_list):
    ax_main.cla();ax_main.set_facecolor('black');ax_main.set_title(f'Galaxy State at Step
{step}',color='white',fontsize=16)

type_color_map={0:"magenta",1:"#333333",2:"yellow",3:"orange",4:"white",5:"cyan",7:"lime",10:"
red",11:"purple"}
    type_size_map={0:200,1:2,2:15,3:10,4:12,5:20,7:25,10:5,11:5}
    unique_types=np.unique(particle_types)
    for p_type in unique_types:
        mask=(particle_types==p_type); marker="*" if p_type==0 else "o"; alpha=0.1 if p_type==1
or p_type>=10 else 1.0

ax_main.scatter(positions[mask,0],positions[mask,1],positions[mask,2],c=type_color_map.get(p
_type),s=type_size_map.get(p_type),alpha=alpha,marker=marker)
    plot_limit=sim_params["GRID_SIZE"]; ax_main.set_xlim(0,plot_limit);
ax_main.set_ylim(0,plot_limit); ax_main.set_zlim(0,plot_limit)
    for axis in [ax_main.xaxis,ax_main.yaxis,ax_main.zaxis]: axis.pane.fill=False;
axis.line.set_color('grey'); axis.set_tick_params(colors='grey')
```

```
    ax_log.cla();ax_log.set_title('Event
Log',color='white');ax_log.set_facecolor('#1a1a1a');ax_log.set_xticks([]);ax_log.set_yticks([])

ax_log.text(0.05,0.95,'\n'.join(event_log_list[-5:]),transform=ax_log.transAxes,color='lightgreen',f
ontsize=10,va='top',family='monospace')
    display(fig); clear_output(wait=True)

# ===          PART 5: DATA SAVING FUNCTION          ===
# =======================================================
def save_final_state(filepath, pos_gpu, vel_gpu, types_gpu, masses_gpu, spin_gpu):
    print(f"\nSaving final simulation state to: {filepath}");
    try:
        os.makedirs(os.path.dirname(filepath), exist_ok=True)
        pos_host=cp.asnumpy(pos_gpu); vel_host=cp.asnumpy(vel_gpu);
types_host=cp.asnumpy(types_gpu)
        masses_host=cp.asnumpy(masses_gpu); spin_host=cp.asnumpy(spin_gpu)
        with h5py.File(filepath, 'w') as f:
            p_group=f.create_group('particles')
            p_group.create_dataset('positions',data=pos_host);
p_group.create_dataset('velocities',data=vel_host)
            p_group.create_dataset('types',data=types_host);
p_group.create_dataset('masses',data=masses_host)
            p_group.create_dataset('spins',data=spin_host)
        print("✅ Save complete.")
    except Exception as e:
        print(f"--- ERROR: Could not save file. ---\n{e}")

# =======================================================
# ===          PART 6: MAIN SIMULATION LOOP          ===
# =======================================================
print(f" Starting Hilios v18 with {total_particles} particles...")
threads_per_block = 256; blocks_per_grid_1d = (total_particles + threads_per_block - 1) //
threads_per_block
event_log = ["Sim Initialized..."]
angular_momentum_history = []
for step in range(sim_params["NUM_STEPS"]):
    merger_log_count_gpu.fill(0)
    merger_kernel((blocks_per_grid_1d,),(threads_per_block,),
(positions_gpu,velocities_gpu,masses_gpu,radii_gpu,is_consumed_gpu,total_particles,merger_l
og_gpu,merger_log_count_gpu,MAX_MERGERS_PER_STEP))
    num_mergers = int(merger_log_count_gpu.get().item())
    if num_mergers > 0:
        merger_events = cp.asnumpy(merger_log_gpu[:num_mergers]); involved_indices =
merger_events.flatten()
```

```
        involved_types = cp.asnumpy(particle_type_gpu[involved_indices]).reshape(num_mergers,
2)
        for i in range(num_mergers):
            survivor_name=label_map.get(involved_types[i,0],"Unk");
consumed_name=label_map.get(involved_types[i,1],"Unk")
            event_log.append(f"MERGER @ {step}: {survivor_name} + {consumed_name}")

    source_term_gpu.fill(0)
    deposit_mass_kernel((blocks_per_grid_1d,),(threads_per_block,),
(positions_gpu,masses_gpu,source_term_gpu,total_particles,gs))
    threads_per_block_3d=(8,8,8); blocks_per_grid_3d=((gs+7)//8,(gs+7)//8,(gs+7)//8)
    update_chronos_kernel((blocks_per_grid_3d,),(threads_per_block_3d,),
(chronos_field_n_plus_1,chronos_field_n,chronos_field_n_minus_1,source_term_gpu,gs,sim_p
arams["DT"],sim_params["K_S"],sim_params["G_RDU"]))
    chronos_field_n_minus_1, chronos_field_n = chronos_field_n, chronos_field_n_plus_1

    b_field_grid.fill(0)

generate_b_field_kernel((blocks_per_grid_3d,),(threads_per_block_3d,),(b_field_grid,positions_
gpu,spin_gpu,particle_type_gpu,total_particles,gs,sim_params["MAGNETAR_STRENGTH"]))

    force_kernel((blocks_per_grid_1d,),(threads_per_block,),
(accelerations_gpu,positions_gpu,velocities_gpu,masses_gpu,particle_type_gpu,chronos_field_
n.ravel(),b_field_grid,total_particles,gs,sim_params["C_ACCRETION_DRAG"],sim_params["AC
CRETION_RADIUS"],sim_params["C_DYN_FRICTION"],sim_params["GAS_CHARGE"],sim_pa
rams["DARK_FORCE_STRENGTH"],sim_params["REPULSION_RADIUS"]))
    velocities_gpu += accelerations_gpu*sim_params["DT"]; positions_gpu +=
velocities_gpu*sim_params["DT"]

    event_flags_gpu.fill(0)
    stellar_evolution_kernel((blocks_per_grid_1d,),(threads_per_block,),
(particle_type_gpu,lifetimes_gpu,event_flags_gpu,total_particles,sim_params["DT"]))

    if cp.any(event_flags_gpu == 1):
        supernova_indices_gpu = cp.where(event_flags_gpu == 1)[0]
        if supernova_indices_gpu.size > 0:
            log_message = f"SUPERNOVA @ {step}: {supernova_indices_gpu.size} star(s)
exploded!"; event_log.append(log_message)
            supernova_positions=cp.asnumpy(positions_gpu[supernova_indices_gpu]);
supernova_masses=cp.asnumpy(masses_gpu[supernova_indices_gpu])

new_particles_pos,new_particles_vel,new_particles_mass,new_particles_type,new_particles_ra
dius,new_particles_lifetimes,new_particles_spins = [],[],[],[],[],[],[]
            for i in range(supernova_indices_gpu.size):
```

```
            pos, original_mass = supernova_positions[i], supernova_masses[i]
            remnant_mass = original_mass * sim_params["REMNANT_MASS_FRACTION"]
            new_particles_pos.append(pos); new_particles_vel.append([0,0,0]);
new_particles_mass.append(remnant_mass)
            if original_mass > 2.5: new_particles_type.append(7)
            else: new_particles_type.append(5)
            new_particles_radius.append(0.01); new_particles_lifetimes.append(1e18);
new_particles_spins.append([0,0,0])
            shell_particles=sim_params["SUPERNOVA_SHELL_PARTICLES"];
ejection_vel=sim_params["SUPERNOVA_EJECTION_VELOCITY"]
            mass_per_shell_particle = (original_mass *
(1.0-sim_params["REMNANT_MASS_FRACTION"]))/shell_particles
            for _ in range(shell_particles):
               theta, phi = np.arccos(2*np.random.rand()-1), np.random.uniform(0,2*np.pi)
               vx,vy,vz = ejection_vel*np.sin(theta)*np.cos(phi),
ejection_vel*np.sin(theta)*np.sin(phi), ejection_vel*np.cos(theta)
               new_particles_pos.append(pos); new_particles_vel.append([vx,vy,vz]);
new_particles_mass.append(mass_per_shell_particle)
               new_particles_type.append(1); new_particles_radius.append(0.001);
new_particles_lifetimes.append(1e18); new_particles_spins.append([0,0,0])
         keep_mask = (event_flags_gpu != 1)

positions_gpu,velocities_gpu,masses_gpu,radii_gpu,is_consumed_gpu,spin_gpu,particle_type_
gpu,lifetimes_gpu =
positions_gpu[keep_mask],velocities_gpu[keep_mask],masses_gpu[keep_mask],radii_gpu[kee
p_mask],is_consumed_gpu[keep_mask],spin_gpu[keep_mask],particle_type_gpu[keep_mask],li
fetimes_gpu[keep_mask]
         num_new_particles = len(new_particles_pos)
         new_pos_gpu=cp.asarray(np.array(new_particles_pos));
new_vel_gpu=cp.asarray(np.array(new_particles_vel));
new_mass_gpu=cp.asarray(np.array(new_particles_mass))
         new_type_gpu=cp.asarray(np.array(new_particles_type),dtype=cp.int32);
new_radius_gpu=cp.asarray(np.array(new_particles_radius));
new_lifetimes_gpu=cp.asarray(np.array(new_particles_lifetimes));
new_spins_gpu=cp.asarray(np.array(new_spins_gpu))
         new_consumed_gpu=cp.zeros(num_new_particles,dtype=cp.int32)
         positions_gpu=cp.vstack([positions_gpu,new_pos_gpu]);
velocities_gpu=cp.vstack([velocities_gpu,new_vel_gpu]);
masses_gpu=cp.hstack([masses_gpu,new_mass_gpu])
         radii_gpu=cp.hstack([radii_gpu,new_radius_gpu]);
is_consumed_gpu=cp.hstack([is_consumed_gpu,new_consumed_gpu]);
spin_gpu=cp.vstack([spin_gpu,new_spins_gpu])
         particle_type_gpu=cp.hstack([particle_type_gpu,new_type_gpu]);
lifetimes_gpu=cp.hstack([lifetimes_gpu,new_lifetimes_gpu])
```

```
        total_particles = positions_gpu.shape[0]; event_flags_gpu = cp.zeros(total_particles,
dtype=cp.int32)

    knoechelman_kernel((blocks_per_grid_1d,), (threads_per_block,),
(positions_gpu,velocities_gpu,masses_gpu,radii_gpu,spin_gpu,particle_type_gpu,is_consumed
_gpu,event_flags_gpu,total_particles,sim_params["A_CONST"],sim_params["N_SENSITIVITY"]
,sim_params["C_UNIFY"],sim_params["K_ROT"],sim_params["C_BALANCE"],sim_params["G_
RDU"],sim_params["TACHYON_MASS"]))
    if cp.any(event_flags_gpu == 2):
        num_events = int(cp.sum(event_flags_gpu == 2).get().item());
event_log.append(f"TRANSFORMATION @ {step}: {num_events} particle(s) converted.")



    if step > 0 and step % sim_params["VIS_INTERVAL"] == 0:
        active_mask = (is_consumed_gpu == 0)
        pos_host=cp.asnumpy(positions_gpu[active_mask]);
vel_host=cp.asnumpy(velocities_gpu[active_mask]);
types_host=cp.asnumpy(particle_type_gpu[active_mask])

        # Calculate Angular Momentum
        if pos_host.shape[0] > 1:
            # Center positions relative to the DRC
            rel_pos = pos_host[1:] - pos_host[0]
            # Get masses for non-DRC particles
            rel_masses = masses_host[1:]

            # Calculate angular momentum for each particle (L = r x p = m * (r x v))
            L_vectors = rel_masses[:, np.newaxis] * np.cross(rel_pos, vel_host[1:])

            # Total angular momentum is the sum of individual vectors
            L_total_vector = np.sum(L_vectors, axis=0)

            # Magnitude of the total angular momentum
            L_total_mag = np.linalg.norm(L_total_vector)
            angular_momentum_history.append(L_total_mag)
        else:
            angular_momentum_history.append(0)


        update_dashboard(step, pos_host, vel_host, types_host, angular_momentum_history,
event_log)

print("✅ Simulation Finished.")
```

```
save_final_state(os.path.join(sim_params["OUTPUT_DIR"],
sim_params["OUTPUT_FILENAME"]), positions_gpu, velocities_gpu, particle_type_gpu,
masses_gpu, spin_gpu)
\end{lstlisting
```