```python
""" Module containing numba kernels for coriolis calculations"""

import numba as nb
import numpy as np
from typing import Tuple, Any

_JIT_OPTIONS = {"nopython": True, "nogil": True, "cache": True}


@nb.jit(**_JIT_OPTIONS)
def _calculate_determinant_y_direction_nb(
    nonhydro: bool,
    nongeo: bool,
    coriolis_dt: np.ndarray,  # Renamed for clarity (dt * strength)
    dChi_full_term: np.ndarray,
) -> np.ndarray:
    """Numba-jitted version: Calculate the determinant.

    Parameters
    ----------
    nonhydro : bool
        The switch between hydrostatic and nonhydrostatic regimes
    nongeo : bool
        The switch between geostrophic and nongeostrophic regimes
    coriolis_dt : np.ndarray
        The coriolis matrix times dt.
    dChi_full_term : np.ndarray
        The derivative of the \bar{Chi} (The mean value of the inverse potential
temperature)
    """
    o1, o2, o3 = coriolis_dt[0], coriolis_dt[1], coriolis_dt[2]
    term1 = (nonhydro - dChi_full_term) * (nongeo**2 + o2**2)
    term2 = nongeo * (o1**2 + o3**2)
    return term1 + term2


@nb.jit(**_JIT_OPTIONS)
def _inverse_matrix_rows_y_direction_nb(
    determinant: np.ndarray,
    nonhydro: bool,
    nongeo: bool,
    coriolis_dt: np.ndarray,
    dChi_full_term: np.ndarray,
) -> Tuple[
    np.ndarray, ...
]:  # Return tuple of 9 arrays (uu, uv, uw, vu, vv, vw, wu, wv, ww)
    """Numba-jitted version: Calculate the inverse extended coriolis matrix when
gravity is on 'y' axis.
    The is matrix in the form (A + Omega + S)
        The components are as follows:
        - A = diag([a_g, a_w, a_g]): The 3x3 diagonal matrix of the switches.
This is due to the LHS.
        - Omega: The usual Coriolis matrix
        - S: The singular matrix as a resulting effect of the buoyancy equation
24d. It only affects on the momentum
             in the direction of the gravity. For y-direction equal to diag([0,
-dt*g*X_y*Theta, 0])
    """

    o1, o2, o3 = coriolis_dt[0], coriolis_dt[1], coriolis_dt[2]
    inv_det = 1.0 / determinant

    ############################################# Row 1:
    term_nh_dchi = nonhydro - dChi_full_term
```

```python
    uu = (nongeo * term_nh_dchi + o1**2) * inv_det
    uv = (nongeo * o3 + o1 * o2) * inv_det
    uw = (o1 * o3 - o2 * term_nh_dchi) * inv_det

    ############################################## Row 2:
    vu = (o1 * o2 - nongeo * o3) * inv_det
    vv = (nongeo**2 + o2**2) * inv_det
    vw = (nongeo * o1 + o2 * o3) * inv_det

    ############################################## Row 3:
    wu = (o1 * o3 + o2 * term_nh_dchi) * inv_det
    wv = (o2 * o3 - nongeo * o1) * inv_det
    ww = (nongeo * term_nh_dchi + o3**2) * inv_det

    return uu, uv, uw, vu, vv, vw, wu, wv, ww


@nb.jit(**_JIT_OPTIONS)
def apply_coriolis_transform_nb_y(
    U_in: np.ndarray,
    V_in: np.ndarray,
    W_in: np.ndarray,
    dChi_full_term: np.ndarray,
    nonhydro: bool,
    nongeo: bool,
    dt: float,
    coriolis_strength: np.ndarray,
) -> Tuple[np.ndarray, np.ndarray, np.ndarray]:
    """
    Numba-jitted core function to apply the inverse Coriolis transform when the
gravity direction in 'y' axis.
    Takes only Numba-compatible types.
    """
    # --- Calculate dt * strength inside ---
    coriolis_dt = dt * coriolis_strength

    # --- Calculate Inverse Matrix Elements ---
    determinant = _calculate_determinant_y_direction_nb(
        nonhydro, nongeo, coriolis_dt, dChi_full_term
    )
    uu, uv, uw, vu, vv, vw, wu, wv, ww = _inverse_matrix_rows_y_direction_nb(
        determinant, nonhydro, nongeo, coriolis_dt, dChi_full_term
    )

    # --- Apply Transform (Matrix-Vector Product) ---
    U_new = uu * U_in + uv * V_in + uw * W_in
    V_new = vu * U_in + vv * V_in + vw * W_in
    W_new = wu * U_in + wv * V_in + ww * W_in

    return U_new, V_new, W_new
```