```python
"""This module is responsible for calculation on the coriolis operator"""

import logging
from typing import TYPE_CHECKING, Union, Any, Tuple, Callable
import numpy as np

if TYPE_CHECKING:
    from atmpy.physics.gravity import Gravity
    from atmpy.variables.variables import Variables
    from atmpy.variables.multiple_pressure_variables import MPV
from atmpy.infrastructure.enums import VariableIndices as VI
from atmpy.time_integrators.coriolis_numba_kernels import
apply_coriolis_transform_nb_y


class CoriolisOperator:
    """
    Encapsulates the handling of coriolis operator in the implicit time update.
This is a part of operator splitting
    for stiff coriolis operator.
    """

    # flag to avoid redundant precompilation of kernels
    _kernel_compiled = False

    def __init__(
        self,
        coriolis_strength: Union[np.ndarray, list, tuple],
        gravity: "Gravity",
    ):
        self.strength: np.ndarray = np.array(coriolis_strength)
        self.gravity: "Gravity" = gravity
        self.coriolis_inverse_kernel: Callable
        self._select_coriolis_kernel()
        self._precompile_numba()

    def _select_coriolis_kernel(self) -> None:
        if self.gravity.direction == "y":
            self.coriolis_inverse_kernel = apply_coriolis_transform_nb_y
        else:
            raise NotImplementedError(
                "The numba kernel for gravity directions other than y are not
implemented."
            )

    def _precompile_numba(self):
        """Attempt to pre-compile the core Numba function."""
        if CoriolisOperator._kernel_compiled:
            logging.debug("Numba Coriolis kernel already compiled/attempted.")
            return
        try:
            print("Pre-compiling Numba Coriolis kernel...")
            # Create dummy data with minimal size but correct types/shapes
            dummy_shape = (2, 2, 2)  # Minimal 3D cell shape
            dummy_momentum = np.zeros(dummy_shape, dtype=np.float64)
            dummy_dchi = np.zeros(dummy_shape, dtype=np.float64)
            dummy_strength = np.zeros(3, dtype=np.float64)  # Coriolis strength
shape

            # Call with dummy data
            _ = self.coriolis_inverse_kernel(
                dummy_momentum.copy(),  # Pass copies if modified in place
                dummy_momentum.copy(),
                dummy_momentum.copy(),
```

```
                dummy_dchi,
                True,  # nonhydro
                True,  # nongeo
                0.1,  # dt
                dummy_strength,
            )
            CoriolisOperator._kernel_compiled = True
            print("Numba Coriolis kernel pre-compiled successfully.")
        except Exception as e:
            print(f"\nWARNING: Could not pre-compile Numba Coriolis kernel.")
            print(f"Error during pre-compilation: {e}")
            print(f"Execution might be slower on the first call.\n")

    def apply_inverse(
        self,
        U: np.ndarray,
        V: np.ndarray,
        W: np.ndarray,
        variables: "Variables",
        mpv: "MPV",
        is_nongeostrophic: bool,
        is_nonhydrostatic: bool,
        Msq: float,
        dt: float,
    ) -> None:
        """Apply correction to momenta due to the coriolis effect. If there are
no coriolis forces in any direction,
        do nothing.

        Parameters
        ----------
        u, v, w: np.ndarray
            momenta components to be updated
        variables : Variables
            The variable container containing the density and temperature
variables.
        mpv : MPV
            The MPV object containing the Chi variable and its derivative
method.
        is_nongeostrophic : bool
            The switch between geostrophic and non-geostrophic regimes.
        is_nonhydrostatic : bool
            The switch between hydrostatic and non-hydrostatic regimes.
        Msq : float
            The mach number squared.
        dt : float
            The time step.
        """

        # Prepare scalar values
        nonhydro = is_nonhydrostatic
        nongeo = is_nongeostrophic
        g: float = self.gravity.strength

        # Calculate dChi (the buoyancy in the momentum equation in the direction
of the gravity)
        dChi = mpv.compute_dS_on_nodes()
        Theta = variables.cell_vars[..., VI.RHOY] / variables.cell_vars[...,
VI.RHO]
        dChi_full_term = (dt**2) * (g / Msq) * dChi * Theta

        # Get elements of the inverse combined matrix (combination of switches,
coriolis matrix and the buoyancy term)
        # See docstrings for more details.
```

```
u_new, v_new, w_new = self.coriolis_inverse_kernel(
    U,
    V,
    W,
    dChi_full_term,
    bool(nonhydro),  # Cast to bool for better handling in Numba
    bool(nongeo),  # Cast to bool for better handling in Numba
    float(dt),  # Cast to float for better handling in Numba
    self.strength,
)

# Update the given variables.
U[...] = u_new
V[...] = v_new
W[...] = w_new
```