Creating Weather Animation Part 2 - Weather Animation with HydroEstimator Data

In [13]:
```
# Importing necessary libraries for handling file operations and data manipulation.

# The glob library is used for collecting file paths that match a specific pattern.
# data files from a dataset, such as HydroEstimator satellite data, need to be acce
from glob import glob

# xarray is a powerful Python library designed to work with labeled multi-dimension
# working with complex data structures often found in satellite data, providing a m
# This library is particularly useful for handling netCDF files commonly used in me
# are formats that the HydroEstimator data might use.
import xarray as xr
```

In [14]:
```
# Importing the os module to interact with the operating system. This module provid
# dependent functionality like reading or writing to the file system.
import os

# Obtain the current working directory where this notebook is running. This is usef
# to the notebook's location, ensuring that the code is portable and can be run on
# modification.
current_directory = os.getcwd()

# Append the subdirectory 'input_data' to the current directory. This is where we e
# Constructing paths in this manner allows for flexibility and easy configuration o
data_directory = os.path.join(current_directory, 'input_data')
```

In [15]:
```
# Generate a list of file paths for data files matching a specific pattern using th

# Concatenate the directory path 'fllst' with the specific pattern for HydroEstimat
# 'NPR.GEO.GHE.v1.*.nc.gz' is designed to match all compressed netCDF files in the
# naming convention. These files contain geospatial data typically used for estimat

# The resulting list 'glst' will contain all file paths that match this pattern, en
# specifying each file's name. This approach is highly efficient for handling large
glst = glob(data_directory + '/NPR.GEO.GHE.v1.*.nc.gz')

# Display the list of file paths to verify correct retrieval and to provide a clear
# This step is essential for debugging and ensuring that the setup correctly captur
glst
```

Out[15]:
```
['c:\\Users\\moham\\OneDrive\\Desktop\\Stevens\\Projects\\FACT\\GitHub\\Module_4
\\input_data\\NPR.GEO.GHE.v1.S202211021530.nc.gz',
 'c:\\Users\\moham\\OneDrive\\Desktop\\Stevens\\Projects\\FACT\\GitHub\\Module_4
\\input_data\\NPR.GEO.GHE.v1.S202211021545.nc.gz',
 'c:\\Users\\moham\\OneDrive\\Desktop\\Stevens\\Projects\\FACT\\GitHub\\Module_4
\\input_data\\NPR.GEO.GHE.v1.S202211021600.nc.gz',
 'c:\\Users\\moham\\OneDrive\\Desktop\\Stevens\\Projects\\FACT\\GitHub\\Module_4
\\input_data\\NPR.GEO.GHE.v1.S202211021615.nc.gz',
 'c:\\Users\\moham\\OneDrive\\Desktop\\Stevens\\Projects\\FACT\\GitHub\\Module_4
\\input_data\\NPR.GEO.GHE.v1.S202211021630.nc.gz',
 'c:\\Users\\moham\\OneDrive\\Desktop\\Stevens\\Projects\\FACT\\GitHub\\Module_4
\\input_data\\NPR.GEO.GHE.v1.S202211021645.nc.gz',
 'c:\\Users\\moham\\OneDrive\\Desktop\\Stevens\\Projects\\FACT\\GitHub\\Module_4
\\input_data\\NPR.GEO.GHE.v1.S202211021700.nc.gz',
 'c:\\Users\\moham\\OneDrive\\Desktop\\Stevens\\Projects\\FACT\\GitHub\\Module_4
\\input_data\\NPR.GEO.GHE.v1.S202211021715.nc.gz',
 'c:\\Users\\moham\\OneDrive\\Desktop\\Stevens\\Projects\\FACT\\GitHub\\Module_4
\\input_data\\NPR.GEO.GHE.v1.S202211021730.nc.gz',
 'c:\\Users\\moham\\OneDrive\\Desktop\\Stevens\\Projects\\FACT\\GitHub\\Module_4
\\input_data\\NPR.GEO.GHE.v1.S202211021745.nc.gz',
 'c:\\Users\\moham\\OneDrive\\Desktop\\Stevens\\Projects\\FACT\\GitHub\\Module_4
\\input_data\\NPR.GEO.GHE.v1.S202211021800.nc.gz',
 'c:\\Users\\moham\\OneDrive\\Desktop\\Stevens\\Projects\\FACT\\GitHub\\Module_4
\\input_data\\NPR.GEO.GHE.v1.S202211021815.nc.gz',
 'c:\\Users\\moham\\OneDrive\\Desktop\\Stevens\\Projects\\FACT\\GitHub\\Module_4
\\input_data\\NPR.GEO.GHE.v1.S202211021830.nc.gz',
 'c:\\Users\\moham\\OneDrive\\Desktop\\Stevens\\Projects\\FACT\\GitHub\\Module_4
\\input_data\\NPR.GEO.GHE.v1.S202211021845.nc.gz',
 'c:\\Users\\moham\\OneDrive\\Desktop\\Stevens\\Projects\\FACT\\GitHub\\Module_4
\\input_data\\NPR.GEO.GHE.v1.S202211021900.nc.gz']
```

In [16]:
```python
# Access the first element (index 0) from the list of file paths (glst).
# In Python, lists are zero-indexed, meaning the first element is accessed with ind
# languages and is crucial for iterating over and accessing elements within data st
fl = glst[0]

# Print the value of the 'fl' variable, which holds the path to the first HydroEsti
# Printing this value serves as a check to ensure that we are retrieving the correc
# It's particularly useful for debugging and verifying that the file paths have bee
print(fl)
```

```
c:\Users\moham\OneDrive\Desktop\Stevens\Projects\FACT\GitHub\Module_4\input_data\NP
R.GEO.GHE.v1.S202211021530.nc.gz
```

In [19]:
```python
# Importing necessary libraries for data handling
import xarray as xr
import os

# Construct the path to the specific HydroEstimator data file you want to access.
# Here, the file is 'NPR.GEO.GHE.v1.Navigation.netcdf.gz' located in the 'input_dat
# First, obtain the current working directory.
current_directory = os.getcwd()

# Append the 'input_data' directory and the specific file name to the current direc
file_path = os.path.join(current_directory, 'input_data', 'NPR.GEO.GHE.v1.Navigatio
```

```python
# Open the dataset using xarray. Assuming the file is compressed in the gzip format
# If xarray's open_dataset cannot directly open '.gz' compressed files, you may nee
ncg = xr.open_dataset(file_path)  # The engine parameter may vary based on file spe

# Display the dataset to verify that it has been loaded correctly.
ncg
```

Out[19]: xarray.Dataset

► Dimensions: (lines: 4800, elems: 10020)

► Coordinates: (0)

▼ Data variables:

latitude	(lines, elems)	float32	...
longitude	(lines, elems)	float32	...

► Indexes: (0)

► Attributes: (0)

In [20]:

```python
# Access and retrieve a specific data point from the 'latitude' variable within the
# This operation demonstrates how to extract individual values from a multi-dimensi

# The 'ncg' dataset contains various geospatial data, including latitude and longit
# is organized in dimensions that typically correspond to different axes in the dat

# Here, we are accessing the latitude value at the first row and first column of th
# so '[0, 0]' refers to the first element in both the row and column dimensions. Th
# to retrieve a specific geographic coordinate for use in further calculations or w

# 'ncg['latitude']' accesses the latitude array within the dataset. Adding '[0, 0]'
# from which to extract the data. The '.data' at the end extracts the actual numeri
# making it usable as a standard Python variable (e.g., for calculations or output)

# It's important to understand this indexing method when working with satellite dat
# necessary for analyzing specific areas or phenomena.
latitude_value = ncg['latitude'][0, 0].data

# Print the extracted latitude value to verify the correct data retrieval and to de
# This step is crucial for debugging and ensuring the accuracy of data manipulation
print("Latitude value at first row and column:", latitude_value)
```

```
Latitude value at first row and column: 64.953
```

In [30]:

```python
# The variable 'fl' is assumed to contain the file path of the NetCDF dataset we in
# It's important to verify that 'fl' is correctly defined and points to a valid Net

# Open a NetCDF dataset using xarray. This library simplifies the process of loadin
# multi-dimensional scientific data. NetCDF (Network Common Data Form) is a widely
# data, especially in meteorology and oceanography.

# 'xr.open_dataset()' is used here to load the NetCDF file specified by the path in
# handles the dataset's dimensions, coordinates, and attributes, providing an acces
```

```
# of the data in Python.
nc = xr.open_dataset(fl)

# Display the dataset object. This will output a summary of the dataset's contents,
# and variables. Displaying the dataset immediately after loading is a good practic
# structure, available variables, and metadata, which are crucial for planning furt
nc
```

Out[30]: xarray.Dataset

► Dimensions: (lines: 4800, elems: 10020)

► Coordinates: (0)

▼ Data variables:

rain	(lines, elems)	float32	...

► Indexes: (0)

► Attributes: (0)

In [31]:

```
# Calculate the longitude and latitude values based on the spatial dimensions of th
# This is important for correctly georeferencing the data in geographical space.

# Retrieve the shape of the 'rain' variable, which represents precipitation data.
# 'ly' corresponds to the number of latitude points, and 'lx' corresponds to the nu
ly, lx = nc['rain'].shape

# Calculate the increment per step in longitude (dtx) across the total range.
# The range here is assumed to be from -180 to 180 degrees. This is calculated by d
# by the number of points in longitude (lx).
dtx = (abs(-180) + abs(180)) / lx

# Generate a list of longitude values starting from -180 degrees, increasing by 'dt
lons = [-180 + (dtx * x) for x in range(lx)]

# Similarly, calculate the increment per step in latitude (dty) from -65 to 65 degr
# The division by 'ly' distributes the latitude values evenly between these bounds.
dty = (abs(-65) + abs(65)) / ly

# Generate a list of latitude values starting from -65 degrees, increasing by 'dty'
lats = [-65 + (dty * y) for y in range(ly)]
lats.reverse()  # Reverse the latitude array if needed, depending on the data orien

# Add longitude and latitude arrays as coordinates to the dataset. This enhances da
# enabling operations that require geographical referencing, such as plotting and s
nc['lon'] = lons
nc['lat'] = lats

# Rename dimensions to be more intuitive. 'lines' are commonly used in image data b
# and similarly, 'elems' is replaced with 'lon'. This renaming makes the dataset di
# which is helpful for subsequent data handling and analysis.
nc = nc.rename({'lines': 'lat', 'elems': 'lon'})
```

```
# Often in meteorological data, a zero value might represent missing or undefined d
# variable with NaN to properly represent missing data. This step is crucial for ac
# as it prevents zero values from skewing analyses and plots.
nc['rain'] = nc['rain'].where(nc['rain'].data != 0)

# Print the updated dataset to verify changes.
nc
```

Out[31]: xarray.Dataset

► Dimensions: (**lat**: 4800, **lon**: 10020)

▼ Coordinates:

lon	(lon)	float64	-180.0 -180.0 ... 179.9 180.0
lat	(lat)	float64	64.97 64.95 64.92 ... -64.97 -65.0

▼ Data variables:

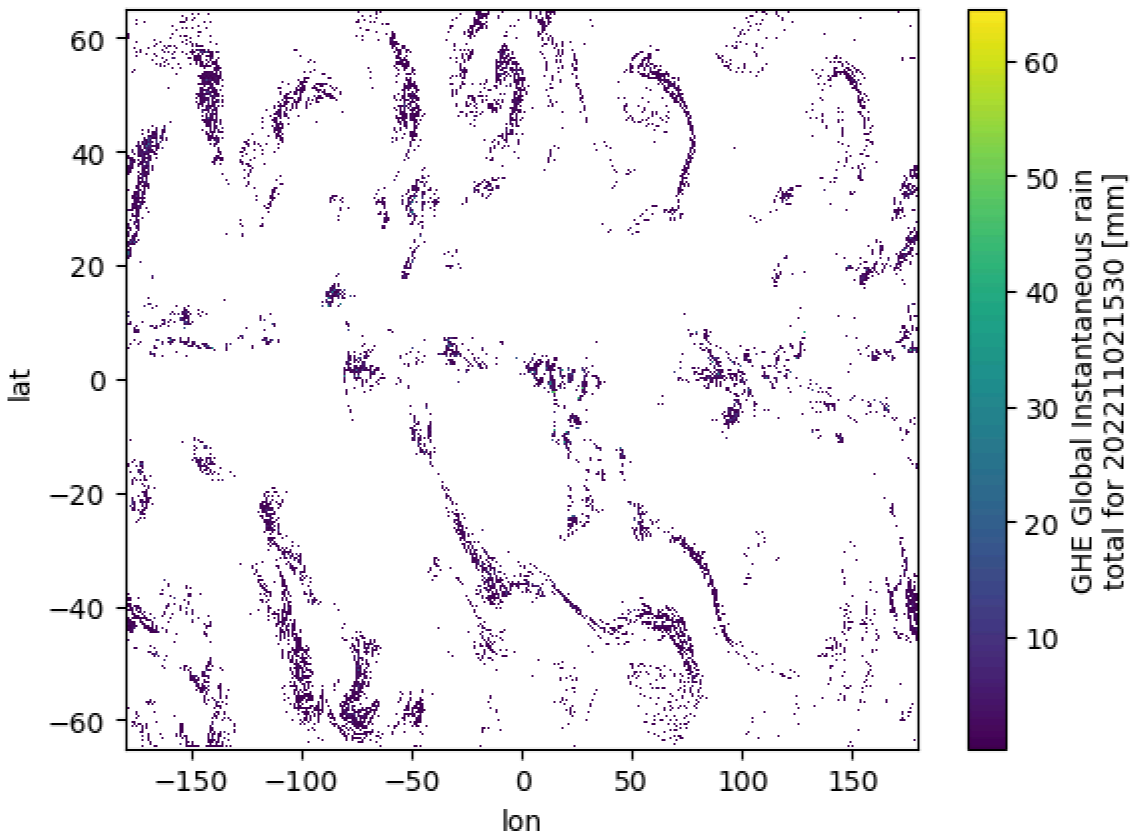
rain	(lat, lon)	float32	nan nan nan nan ... nan nan nan nan

► Indexes: (2)

► Attributes: (0)

In [33]:
```
# Plot the 'rain' variable from the dataset using xarray's built-in plotting capabi
# Xarray integrates with Matplotlib to provide a convenient way to visualize data d
# This is particularly useful for quick examinations of data and preliminary analys

# 'nc['rain'].plot()' automatically creates a plot of the 'rain' variable. This fun
# labelled data, automatically selecting the appropriate plot type (in this case, l
# It labels axes based on the dimensions of the data and uses the variable's metada

# This kind of plot is essential for initial data exploration and quality control,
# identify patterns, anomalies, or issues within the dataset. For example, visualiz
# areas of heavy rainfall or comparing observed patterns against meteorological pre

nc['rain'].plot()

# Additional customizations can be added to enhance the plot. For example, adding g
# or adjusting the color scheme. These can be done using additional arguments in th
# the Matplotlib axes object that the plot function returns.

# Example of customizing the plot:
# ax = nc['rain'].plot()
# ax.set_title('Rainfall Intensity')
# ax.set_xlabel('Longitude')
# ax.set_ylabel('Latitude')
# ax.grid(True)
```

Out[33]: `<matplotlib.collections.QuadMesh at 0x19c277f72f0>`



In [14]:
```
# Select a subset of the dataset based on longitude and latitude slices
ds = nc.sel(lon=slice(-123, -74.5), lat=slice(37, 10))
ds
```

Out[14]: xarray.Dataset

► Dimensions: (**lat**: 997, **lon**: 1350)

▼ Coordinates:

lon	(lon)	float64	-123.0 -122.9 ... -74.55 -74.51
lat	(lat)	float64	37.0 36.97 36.94 ... 10.05 10.02

▼ Data variables:

rain	(lat, lon)	float32	nan nan nan nan ... nan nan nan nan

► Indexes: (2)

► Attributes: (0)

In [34]:
```
# Selecting a specific geographic subset from the dataset based on longitude and la
# This operation is crucial in spatial data analysis, especially when the focus is 
# dealing with large datasets where reducing the area of interest can significantly

# The 'nc' dataset contains comprehensive global or regional weather data, but for 
# you might only need data from a particular area. Using xarray's .sel() method all
# for coordinates (in this case, longitude and latitude) to extract only the data r
```

```
# Here, we define longitude slices from -123 to -74.5 and latitude slices from 37 t
# effectively narrowing down the dataset to cover a specific part of the Western He
# likely focusing on significant portions of North and Central America.

# The 'slice' function is used to define the start and end points for each dimensio
# which xarray uses to select the corresponding range from the dataset.
ds = nc.sel(lon=slice(-123, -74.5), lat=slice(37, 10))

# Display the newly selected subset of the dataset.
# This output allows us to verify that the dimensions and variable sizes reflect th
# ensuring that the selection was performed correctly. It's an essential step for c
# and appropriateness for subsequent analysis.
print(ds)

# Further operations can now be performed on this subset, such as detailed data ana
# or exporting to a different format for use in other applications or reports.
```

```
<xarray.Dataset> Size: 5MB
Dimensions:  (lat: 997, lon: 1350)
Coordinates:
  * lon      (lon) float64 11kB -123.0 -122.9 -122.9 ... -74.59 -74.55 -74.51
  * lat      (lat) float64 8kB 37.0 36.97 36.94 36.91 ... 10.1 10.08 10.05 10.02
Data variables:
    rain     (lat, lon) float32 5MB nan nan nan nan nan ... nan nan nan nan nan
```

In [35]:
```
# Import necessary libraries for plotting and mapping
import matplotlib.pyplot as plt  # Matplotlib for plotting
import matplotlib.colors as mcolors  # Matplotlib colors for colormap
import cartopy  # Cartopy for geographic projections
import cartopy.crs as ccrs  # Cartopy coordinate reference systems
```

In [36]:
```
# Define a custom color map for precipitation data visualization.
# Each tuple in the list represents an RGB color.
cmap_data = [
    (1.0, 1.0, 1.0),  # white
    (0.3137255012989044, 0.8156862854957581, 0.8156862854957581),  # light cyan
    (0.0, 1.0, 1.0),  # cyan
    (0.0, 0.8784313797950745, 0.501960813999176),  # aquamarine
    (0.0, 0.7529411911964417, 0.0),  # green
    (0.501960813999176, 0.8784313797950745, 0.0),  # yellow-green
    (1.0, 1.0, 0.0),  # yellow
    (1.0, 0.6274510025978088, 0.0),  # orange
    (1.0, 0.0, 0.0),  # red
    (1.0, 0.125490203499794, 0.501960813999176),  # magenta
    (0.9411764740943909, 0.250980406999588, 1.0),  # purple
    (0.501960813999176, 0.125490203499794, 1.0),  # dark purple
    (0.250980406999588, 0.250980406999588, 1.0),  # indigo
    (0.125490203499794, 0.125490203499794, 0.501960813999176),  # dark blue
    (0.125490203499794, 0.125490203499794, 0.125490203499794),  # black
    (0.501960813999176, 0.501960813999176, 0.501960813999176),  # gray
    (0.8784313797950745, 0.8784313797950745, 0.8784313797950745),  # light gray
    (0.9333333373069763, 0.8313725590705872, 0.7372549176216125),  # beige
    (0.8549019694328308, 0.6509804129600525, 0.47058823704719543),  # brown
    (0.6274510025978088, 0.42352941632270813, 0.23529411852359772),  # dark brown
    (0.4000000059604645, 0.2000000298023224, 0.0)  # deep brown
]
```

```
# Define the levels of precipitation to classify the data into different ranges.
# These levels are used for normalizing color representation based on precipitation
clevs = [0, 1, 2.5, 5, 7.5, 10, 15, 20, 30, 40, 50, 70, 100, 150, 200, 250, 300, 40

# Create a colormap object using the defined color map data and name it 'precipitat
cmap = mcolors.ListedColormap(cmap_data, 'precipitation')

# Create a BoundaryNorm object for normalization.
# It maps the precipitation values (clevs) into discrete intervals for the colormap
norm = mcolors.BoundaryNorm(clevs, cmap.N)
```

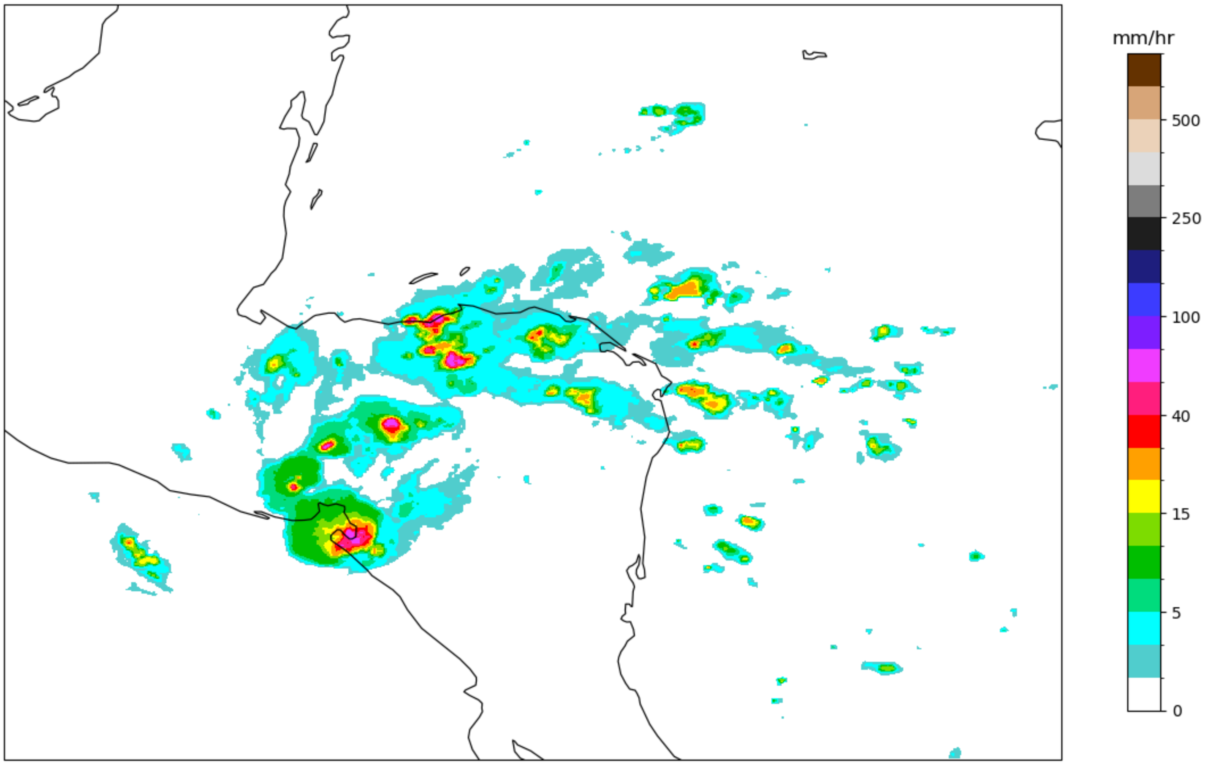
In [37]:

```
# Create a figure with a specified size (15x15 inches) for plotting.
fig = plt.figure(figsize=(15, 15))

# Define the map projection as 'PlateCarree' (geographical coordinates).
proj = ccrs.PlateCarree()

# Add a subplot to the figure with the specified projection.
ax = fig.add_subplot(1, 1, 1, projection=proj)

# Display the 'rain' data from the dataset as an image on the axes.
# Setting the extent of the image to match the geographical coordinates in the data
# The colormap and normalization are applied to represent the rain intensity.
cf = ax.imshow(
    ds['rain'].data,
    extent=(
        ds['lon'].min().data,
        ds['lon'].max().data,
        ds['lat'].min().data,
        ds['lat'].max().data
    ),
    cmap=cmap,
    norm=norm,
    transform=proj
)

# The commented line below represents an alternative method using contour filling.
# ax.contourf(ds['lon'], ds['lat'], ds['rain'].data, clevs, cmap=cmap, norm=norm)

# Add coastlines to the map for better geographical reference.
# Setting resolution to '50m' and line color to black with a linewidth of 0.85.
ax.coastlines(resolution='50m', color='black', linewidth=0.85)

# Set the extent of the map in longitude and latitude.
# This defines the visible area of the map.
ax.set_extent([-92.0, -78.0, 10.0, 20.0])

# Add a colorbar to the figure for reference.
# 'shrink' controls the size of the colorbar.
# Set the title of the colorbar to 'mm/hr' to indicate the unit of rainfall intensi
cb = plt.colorbar(cf, shrink=0.5)
cb.ax.set_title('mm/hr')
```

Out[37]:

```
Text(0.5, 1.0, 'mm/hr')
```

```
c:\Users\moham\miniconda3\Lib\site-packages\cartopy\io\__init__.py:241: DownloadWarn
ing: Downloading: https://naturalearth.s3.amazonaws.com/50m_physical/ne_50m_coastlin
e.zip
  warnings.warn(f'Downloading: {url}', DownloadWarning)
```

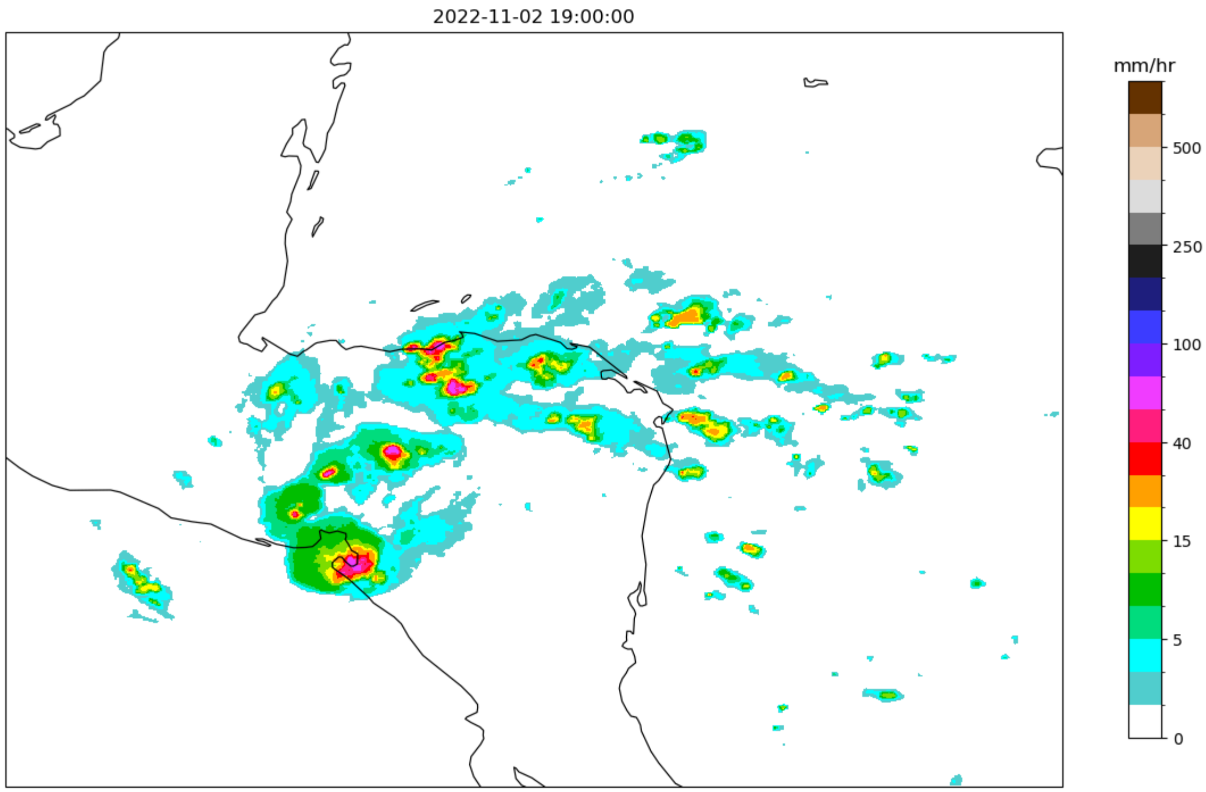


In [38]:
```python
import os # Import the os module
from datetime import datetime  # Import datetime class from datetime module

# Parse a specific string format into a datetime object
ddte = datetime.strptime('S202211021900', "S%Y%m%d%H%M")
```

In [21]:
```python
# Set up a 15x15 inch figure for plotting
fig = plt.figure(figsize=(15,15))

# Use PlateCarree projection for geographic map plotting
proj = ccrs.PlateCarree()

# Add a subplot to the figure with the specified map projection
ax = fig.add_subplot(1, 1, 1, projection=proj)

# Display 'rain' data as an image on the map with the specified color mapping and n
cf = ax.imshow(ds['rain'].data,
               extent=(ds['lon'].min().data, ds['lon'].max().data,
                       ds['lat'].min().data, ds['lat'].max().data),
               cmap=cmap, norm=norm, transform=proj)

# Alternatively, uncomment to use contour filling for 'rain' data visualization
# cf = ax.contourf(ds['lon'], ds['lat'], ds['rain'].data, clevs, cmap=cmap, norm=no

# Add coastlines to the map for better reference, with specified resolution and sty
ax.coastlines(resolution='50m', color='black', linewidth=0.85)
```

```
# Define the geographical extent of the map
ax.set_extent([-92.0, -78.0, 10.0, 20.0])

# Set the title of the plot using the datetime object 'ddte'
plt.title(ddte)

# Add a colorbar to the plot, with a title representing the unit of measurement
cb = plt.colorbar(cf, shrink=0.5)
cb.ax.set_title('mm/hr')
```

Out[21]: Text(0.5, 1.0, 'mm/hr')



In [22]:
```
import numpy as np  # Import NumPy for numerical operations and array handling
import matplotlib.pyplot as plt  # Import Pyplot for plotting and visualization
from matplotlib import animation # Import animation module for creating animations
from IPython.display import HTML  # Import HTML to display HTML content in Jupyter
```

In [24]:
```
def read_hyest(fl):
    # Open the dataset from the file
    nc = xr.open_dataset(fl)

    # Get the shape of the 'rain' data to calculate latitudes and longitudes
    ly, lx = nc['rain'].shape

    # Calculate longitude values based on dataset shape
    dtx = (abs(-180) + abs(180)) / lx
    lons = [-180 + (dtx * x) for x in range(lx)]

    # Calculate latitude values based on dataset shape
    dty = (abs(-65) + abs(65)) / ly
    lats = [-65 + (dty * y) for y in range(ly)]
    lats.reverse()  # Reverse latitudes as they are typically from North to South
```

```
    # Assign calculated longitude and latitude values to the dataset
    nc['lon'] = lons
    nc['lat'] = lats

    # Rename dimensions for clarity
    nc = nc.rename({'lines': 'lat', 'elems': 'lon'})

    # Uncomment below to filter out zero values in 'rain' data
    # nc['rain'] = nc['rain'].where(nc['rain'].data != 0)

    return nc
```

In [25]:

```
import cartopy.crs as ccrs  # Import Cartopy's coordinate reference systems for map
import cartopy.feature as cfeature  # Import Cartopy's feature module for adding ge
```

In [26]:

```
# Set up a 10x10 inch figure for plotting
fig = plt.figure(figsize=(10,10))

# Use PlateCarree projection for geographic map plotting
proj = ccrs.PlateCarree()

# Add a subplot to the figure with the specified map projection
ax = fig.add_subplot(1, 1, 1, projection=proj)

# Add coastlines to the map for better reference
ax.coastlines(resolution='50m', color='black', linewidth=0.85)

# Define the geographical extent of the map
ax.set_extent([-92.0, -78.0, 10.0, 20.0])

# Initialize a list to store image frames for animation
ims = []

# Loop through each file in the list 'glst'
for n, fl in enumerate(glst):
    # Read dataset using the custom function 'read_hyest'
    nc = read_hyest(fl)

    # Subset the dataset for the desired longitude and latitude range
    ds = nc.sel(lon=slice(-123, -74.5), lat=slice(37, 10))

    # Process the 'rain' data for the current frame
    d0 = ds['rain']
    dd = d0 if n == 0 else dd + d0

    # Plot the data as an image and add to the frame list
    im = ax.imshow(dd.data, extent=(ds['lon'].min().data, ds['lon'].max().data, ds[
                   cmap=cmap, norm=norm, transform=proj)
    frame = [im]
    ims.append(frame)

# Prevent display of static plot
plt.close()
```

```python
# Create the animation
ani = animation.ArtistAnimation(fig, ims)

# Render the animation in the notebook as an interactive JavaScript widget
HTML(ani.to_jshtml())
```

Out[26]: No description has been provided for this image

Once Loop Reflect

In [27]:
```python
# Initialize a figure with a specified size
fig = plt.figure(figsize=(10,10))

# Set the map projection to PlateCarree (equidistant cylindrical projection)
proj = ccrs.PlateCarree()

# Add a subplot with the specified projection
ax = fig.add_subplot(1, 1, 1, projection=proj)

# Draw coastlines for reference, with a specified resolution and style
ax.coastlines(resolution='50m', color='black', linewidth=0.85)

# Set the geographic extent of the map
ax.set_extent([-92.0, -78.0, 10.0, 20.0])

# Initialize a list to hold frames for animation
ims = []

# Loop through each file in the list 'glst'
for n, fl in enumerate(glst):
    frame = []

    # Read the dataset from the file using the custom function
    nc = read_hyest(fl)

    # Subset the dataset for a specific longitude and latitude range
    ds = nc.sel(lon=slice(-123, -74.5), lat=slice(37, 10))

    # Extract the 'rain' data from the dataset
    d0 = ds['rain']

    # Accumulate the 'rain' data over iterations or start new for the first iterati
    if n == 0:
        dd = d0.copy()
    else:
        dd += d0

    # Create an image from the 'rain' data and add it to the current frame
    im = ax.imshow(dd.data,
                   extent=(ds['lon'].min().data, ds['lon'].max().data,
                           ds['lat'].min().data, ds['lat'].max().data),
                   cmap=cmap, norm=norm, transform=proj)
```

```
    frame.append(im)
    ims.append(frame)

# Close the plt.show() window to prevent display of a static plot
plt.close()

# Create an animation from the frames
ani = animation.ArtistAnimation(fig, ims)

# Display the animation as an interactive JavaScript widget in Jupyter Notebook
HTML(ani.to_jshtml())
```

Out[27]: No description has been provided for this image

Once Loop Reflect