"""

GPU Acceleration for Temporal Flow Computations

=============================================

"""

import torch

import torch.nn as nn

from torch.utils.data import DataLoader

class GPUAccelerator:

def \_\_init\_\_(self, device='cuda'):

self.device = torch.device(device if torch.cuda.is\_available() else 'cpu')

self.precision = torch.float32

def setup\_computation(self, problem\_size):

"""

Setup GPU computation parameters

"""

# Determine optimal chunk size for GPU memory

self.chunk\_size = self.optimize\_chunk\_size(problem\_size)

# Initialize CUDA streams for overlap

self.streams = [torch.cuda.Stream() for \_ in range(3)]

def optimize\_chunk\_size(self, problem\_size):

"""

Determine optimal chunk size based on GPU memory

"""

gpu\_mem = torch.cuda.get\_device\_properties(0).total\_memory

return min(problem\_size, gpu\_mem // (self.precision.itemsize \* 3))

@torch.cuda.amp.autocast()

def compute\_field\_evolution(self, field\_data):

"""

Compute field evolution using GPU acceleration

"""

# Move data to GPU

field\_tensor = torch.from\_numpy(field\_data).to(self.device)

# Compute in chunks

results = []

for chunk in torch.split(field\_tensor, self.chunk\_size):

with torch.cuda.stream(self.streams[0]):

# Spatial derivatives

derivatives = self.compute\_derivatives\_gpu(chunk)

with torch.cuda.stream(self.streams[1]):

# Forces

forces = self.compute\_forces\_gpu(chunk)

with torch.cuda.stream(self.streams[2]):

# Temporal evolution

evolution = self.temporal\_step\_gpu(chunk, derivatives, forces)

results.append(evolution)

# Synchronize streams

torch.cuda.synchronize()

return torch.cat(results).cpu().numpy()

class CUDAKernels:

"""

Custom CUDA kernels for critical computations

"""

@staticmethod

def compile\_kernels():

"""

Compile custom CUDA kernels

"""

kernel\_source = """

extern "C" {

\_\_global\_\_ void temporal\_flow\_kernel(float\* W, float\* dW, float\* result,

int N, float dt, float dx) {

int idx = blockIdx.x \* blockDim.x + threadIdx.x;

if (idx < N) {

// Implement core temporal flow computation

// ...

}

}

}

"""

return cupy.RawKernel(kernel\_source, 'temporal\_flow\_kernel')

"""

Usage Example:

-------------

# Initialize GPU accelerator

accelerator = GPUAccelerator()

accelerator.setup\_computation(problem\_size=1e6)

# Compile custom kernels

kernels = CUDAKernels.compile\_kernels()

# Perform GPU-accelerated computation

field\_data = np.random.rand(1000000)

result = accelerator.compute\_field\_evolution(field\_data)

"""