中国科学院大学 2025 秋《GPU 架构与编程》大作业一：SCNN 加速推理系统

1. 简介（Intro）

本项目是中国科学院大学 2025 秋季课程《GPU 架构与编程》大作业一，实现并加速一个基于脉冲卷积神经网络（Spiking CNN, SCNN）的 Fashion-MNIST 分类系统。
项目包含：

- **训练端（PyTorch + SpikingJelly）**：在 CPU / Apple MPS / CUDA 上完成 SCNN 训练，并导出权重为纯文本。
- **推理端（CUDA C++）**：在 GPU 上使用手写 CUDA Kernel（含 PTX 优化版）进行高性能 SNN 推理与性能评测。

2. 目录结构

- `train.py`：基于 SpikingJelly 的 SCNN 训练脚本，支持离线使用本地 Fashion-MNIST 数据。
- `inference.cu`：CUDA 推理主程序，使用高性能 Kernel（共享内存、向量化、CUDA Graph 等）完成 SNN 推理。
- `inference_ptx.cu`：在 `inference.cu` 基础上，进一步引入显式 PTX 指令（如 `ld.global.ca`、`fma.rn`）的优化版本。
- `data/`
  - `fashion/`：原始 Fashion-MNIST `*.gz` 文件（通过 `train.py` 自动放置到 `data/FashionMNIST/raw`）。
- `*.weight.txt / *.bias.txt`：训练完成后导出的各层权重与偏置，用于 GPU 推理。

3. 项目特色

- **端到端 SNN 加速推理链路**
  - `train.py` 使用 SpikingJelly 构建多层卷积 + IF 神经元的 SCNN，并与 GPU 端的网络结构一一对应。
  - 训练结束后将 `conv1/conv2/fc1/fc2/fc3` 的权重/偏置展平为 `.txt`，由 CUDA 程序直接加载，避免框架依赖。
- **高性能 CUDA 推理（`inference.cu`）**
  - **卷积 + IF 融合 Kernel**：将卷积、IF 神经元更新、脉冲发放合并在一个 Kernel 内，减少中间特征图访存。
  - **共享内存 Tile 优化**：`conv2d_if_smem5x5_tiled_kernel` 等 Kernel 利用共享内存缓存局部输入 Patch，提升卷积算子数据复用。
  - **向量化 FC 计算**：全连接层使用 `float4` 加载与 `fmaf` 指令进行向量化累加，提高吞吐。
  - **U8 稀疏输入与脉冲编码**：第一层池化输出以 `unsigned char`（0/1）形式存储，第二层卷积基于稀疏脉冲输入做条件累加，显著降低带宽与乘加次数。
  - **CUDA Graph + 双缓冲 Pipeline**：使用 CUDA Graph 捕获批次计算图，并在 H2D/D2H 与计算之间做流级重叠，提高整体吞吐。

- **显式 PTX 优化（`inference_ptx.cu`）**
  - 在关键算子（卷积、全连接）中显式使用 PTX 指令：
    - `ld.global.ca.f32 / ld.global.ca.u8`：带 cache hint 的全局加载，提升访存局部性。
    - `fma.rn.f32`：显式 FMA 指令，减少舍入误差并提高算术吞吐。
  - 提供对比基线：`inference.cu`（纯 CUDA C 实现） vs `inference_ptx.cu`（内嵌 PTX），用于分析 PTX 级微优化带来的性能差异。

4. 运行方式

4.1 环境准备

- 训练端：
  - Python 3.9+
  - `torch`, `torchvision`, `spikingjelly`, `numpy`
- 推理端：
  - 支持 CUDA 的 GPU + CUDA Toolkit（包含 `nvcc`）

4.2 离线准备 Fashion-MNIST 数据

将官方 Fashion-MNIST 四个压缩文件放在：

- `data/fashion/train-images-idx3-ubyte.gz`
- `data/fashion/train-labels-idx1-ubyte.gz`
- `data/fashion/t10k-images-idx3-ubyte.gz`
- `data/fashion/t10k-labels-idx1-ubyte.gz`

`train.py` 会自动将它们链接/拷贝到 `data/FashionMNIST/raw/` 并用 `torchvision.datasets.FashionMNIST` 生成 `processed/*.pt`。

4.3 训练 SCNN 并导出权重

```
cd /Users/unit-a/Desktop/GPU
python train.py
```

训练结束后，当前目录下会生成：

- `conv1.weight.txt`, `conv1.bias.txt`
- `conv2.weight.txt`, `conv2.bias.txt`
- `fc1.weight.txt`, `fc1.bias.txt`
- `fc2.weight.txt`, `fc2.bias.txt`
- `fc3.weight.txt`, `fc3.bias.txt`

这些文件会被 GPU 推理程序直接加载。

4.4 编译与运行 GPU 推理

以 `inference.cu` 为例：

```
cd /Users/unit-a/Desktop/GPU
nvcc -O3 -arch=sm_80 inference.cu -o inference    # 根据实际 GPU 修改 -arch
./inference .                                     # 参数为模型与权重所在目录
```

使用 PTX 优化版本：

```
nvcc -O3 -arch=sm_80 inference_ptx.cu -o inference_ptx
./inference_ptx .
```

程序会输出：

```
<总耗时秒数>:<测试集准确率>
```

例如：

```
0.0234:0.9050
```

5. 致谢

- 感谢中国科学院大学《GPU 架构与编程》课程组提供的大作业框架与评测平台。
- 感谢 PyTorch / TorchVision / SpikingJelly 等开源社区，为本项目训练端提供了可靠的工具链。
- 感谢 NVIDIA CUDA 工程文档与社区示例，为本项目的内存优化、CUDA Graph 与 PTX 调优提供了重要参考。