PART-A: Tiled Matrix Multiply on CPU

Learning goal: The importance of tiling and data reuse on performance of matrix multiplication

You will write a hand-optimized matrix multiplication kernel for C = A × B:

1. Start with a naïve triple loop.
2. Add blocking/tiling.
3. Measure speedup and memory traffic.

Assume that the tile is sized such that it fits on the "on-chip SRAM" and misses are served by DRAM. While your code will run on a CPU, it will still help you reason like hardware designers and understand why typical ML accelerators have compute co-located with SRAM.

What you implement

A1 Simple matrix kernel (naïve):

A2 Blocked kernel with a manual software-managed buffer:

- Pick **tile sizes of 128x128**
- Operate on tile of A and B (assume they are transported into on-chip "SRAM").
- Multiply-accumulate tiles before evicting them.

Analysis:

- Do your experiments for N=256, 512, 1024, 2048 and 4096
- Time both version simple and tiled kernels
- Count total bytes read/written from "DRAM" vs from "SRAM"
- What is the arithmetic intensity of the kernel as N is varied

---

What you submit for Part-A

- Code for naïve and tiled versions.
- Runtime (ms) for each version on multiple matrix sizes (e.g. 256, 512, 1K, 2K, 4K).
- Achieved GFLOP/s = $(2 \cdot N^3$ ops) / runtime.
- Effective DRAM bandwidth used (bytes/time).
- A one-page writeup answering:
  - Why does tiling help so much?
  - What is the size of "SRAM tile" needed for tiled kernel?
  - If you had *many* MAC units in parallel, what becomes the bottleneck?

PART-B: Simulated Systolic Array for GEMM

Goal: To understand how systolic array (output-stationary) works.

You will build a tiny cycle-level simulator of a 2D grid of MAC units (say 4×4). Each MAC:

- Receives an activation from the left each cycle,
- Receives a weight from the top,
- Multiplies, accumulates, forwards.

This is a classic systolic array (output stationary). We have provided the code for FIFO, and most of the code for PE.

What you need to do

1. **Model:**
   - Fill the **PE_Cycle** function
   - Fill the **Sysarray_Init** and **Sysarray_Cycle** functions
2. **Driver:**
   - We provide the main.c driver that contains A and B.
   - Run the simulator cycle by cycle.
   - Record:
     - Total cycles to compute an N×N multiply.
     - What is the arithmetic intensity of this computation (bytes/cyle) assume each element is 1 byte
3. **Experiments:**
   - Vary array size (N): 4×4 vs 16×16 vs 64×64 vs 256x256.
   - Plot the relationship of cycles to finish computation versus N for NxN
   - Plot throughput (MACs per cycle delivered / MACs per cycle peak).
   - Plot off-chip bandwidth demand versus N.

Deliverables

- Simulator code only for **pe.c** and **systolic.c** (fifo.c should remain unchanged and your code should work with our main.c)
- Plots of:
  - Latency vs array size.
  - Bandwidth requirements vs array size.
- A one-page writeup:
  - Explaining the arithmetic intensity versus N
  - The size requirement of SRAM buffers (FIFO queues) as N is varied
  - The bandwidth requirement as N is varied