MCU Memory Hierarchy

Given the limited resources available on microcontroller units (MCUs), it is paramount for us as future tinyML engineers to understand the different types of on-device memory, including their interplay and performance capabilities. This understanding will become vital for optimizing machine learning models on a given hardware architecture.

What is memory?

Computer memory stores information as binary values, known as **bits**, which can take on a value or one or zero. However, storing a single bit is not particularly informative, and so multiple bits are often grouped together into quantities known as **bytes**, which consist of 8 bits. A byte can represent 256 distinct values, which can be used to represent anything from letters of the alphabet to the weights of a neural network.

Microcontrollers read from and write to memory using bytes, which are located by a **memory address**. This memory address is a hexadecimal value that tells the microcontroller where the memory it is looking for is located, much like a postal address. Microcontrollers have a limited amount of memory, which fundamentally limits the complexity of programs and computations they can run. Different forms of memory may also take longer to access or lose data when they are not constantly powered, leading to the use of multiple forms of memory for various purposes.

RAM and Flash Memory

The two most important forms of memory are **flash memory** and **random access memory (RAM)**. Our Arduino Nano 33 BLE Sense is equipped with 1 MB of flash memory and 256 KB of static RAM.

Flash memory is **non-volatile**, meaning that data persists even when the system is turned off. This memory is where we save our model weights and program code, and is what we are transferring when we “flash” our model on to the device. The process of saving to flash memory is slow and gradually degrades the memory over time. Consequently, to minimize this degradation, flash memory is effectively used as **read-only memory** and only overwritten during reprogramming, not when storing intermediate results during program execution.

RAM, however, is **volatile**, and thus data is lost when the device is powered off. Accordingly, RAM is used for the **temporary storage** of variables, such as input and output buffers, and intermediate tensors. RAM is much faster to read and write than flash memory, making it ideal as the primary memory used during program execution. RAM comes in two forms: static and dynamic.

Dynamic RAM (DRAM) uses a single transistor and capacitor to store a bit, but the capacitor quickly loses charge and must be periodically refreshed to prevent the stored information from being lost. **Static RAM** (SRAM) uses 6 transistors to store each bit, but is able to maintain the stored information without refreshing. SRAM is more expensive due to the additional transistors needed, but is faster and less power-intensive because no time or energy is spent performing a DRAM refresh between read and write operations. These differences make DRAM a suitable candidate for main memory in modern computers, and SRAM more suitable for caches. In our microcontroller, we have access to SRAM and use it as our main memory during program execution.

A third type of memory is also used in microcontrollers but is often abstracted from the user; this type of memory is known as a **register**. Typically, special purpose registers exist that store various values needed to perform certain low-level computing functions, such as the program counter and stack pointer. In contrast, general purpose registers are used to store values and memory addresses. It is unlikely you will need to play around with registers as a tinyML engineer unless you are working at the assembly code level.

Relevance to TF Lite Micro

Why do we care about introducing computer memory to you? Well, the size of your model weights and code cannot exceed the size of the available MCU flash memory (i.e., flash memory minus the 2 KB bootloader), and the size of input buffers and intermediate tensors cannot exceed the available SRAM on the device. These are hard constraints, and as tinyML engineers we must be wary when working with large models that might exceed these memory constraints. In such cases, an MCU platform with additional resources or a simpler model may be necessary.