tinyML® Research Symposium

Enabling Ultra-low Power Machine Learning at the Edge

March 27, 2023

www.tinyML.org

MEMA Runtime Framework

Minimizing External Memory Accesses for TinyML Devices

Andrew Sabot
Vikas Natesh

Why TinyML?

Tiny machine learning (TinyML) applications include person detection, keyword spotting, and anomaly prediction.

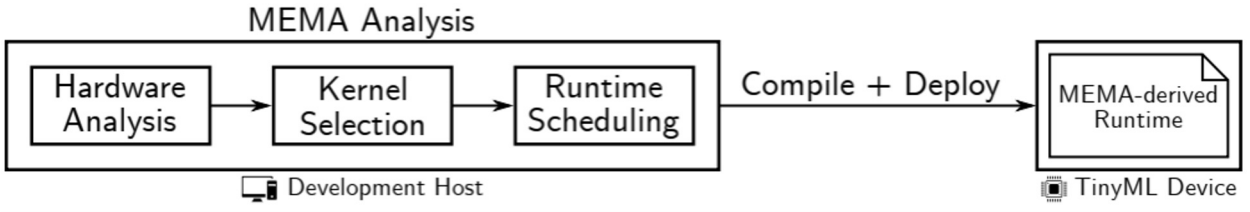
Devices ranging from industrial sensors to mobile phones can benefit from:

- Lower **latency** due to local computation
- Increased privacy as data is kept local
- Decreased reliance on network connectivity
- Improved **energy consumption** from reducing external communication

Screenshot from *tinyML Talks: Constrained Object Detection on Microcontrollers with FOMO*

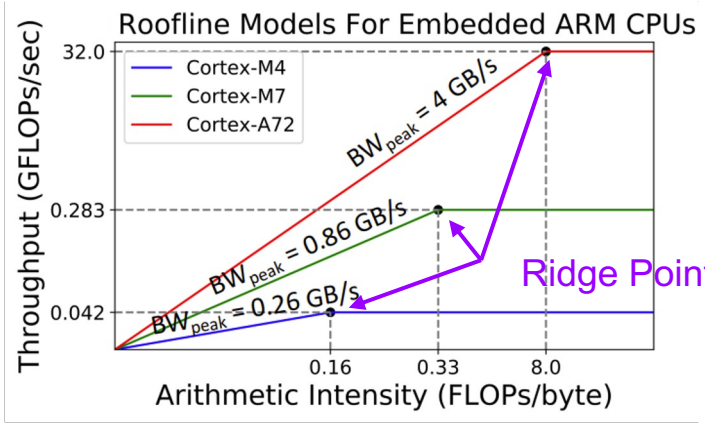
MEMA Overview

- The MEMA Runtime Framework aims to facilitate the deployment of efficient **inference runtimes** on power-constrained devices
- MEMA considers hardware limitations (e.g., local memory, register count) and problem sizes when analytically deriving runtime schedules that minimize data movement
- We illustrate the framework using matrix multiplication (MM), which is commonly used in neural networks (e.g., for convolutions)



Characterizing Hardware

- Roofline model defined by hardware characteristics of each platform:
  - Peak throughput (e.g., FP32 operations)
  - Peak memory bandwidth
- Two performance regimes:
  - Memory-bound (left of ridge point)
  - Compute-bound (right of ridge point)
- Benefits of reducing IO for these regimes via runtime scheduling:
  - Memory-bound: increase throughput
  - Compute-bound: reduce energy consumption



MEMA Analysis of MM for Hardware

For MEMA analysis, we:

1. Identify device characteristics (e.g., memory hierarchy and register count)
2. Enumerate possible **tile sizes** and **streaming strategies**
3. Determine which kernels are applicable to the tile sizes
4. Using tile sizes and kernel choices, evaluate **loop order** IO requirements
5. Select loop order with the lowest IO requirements

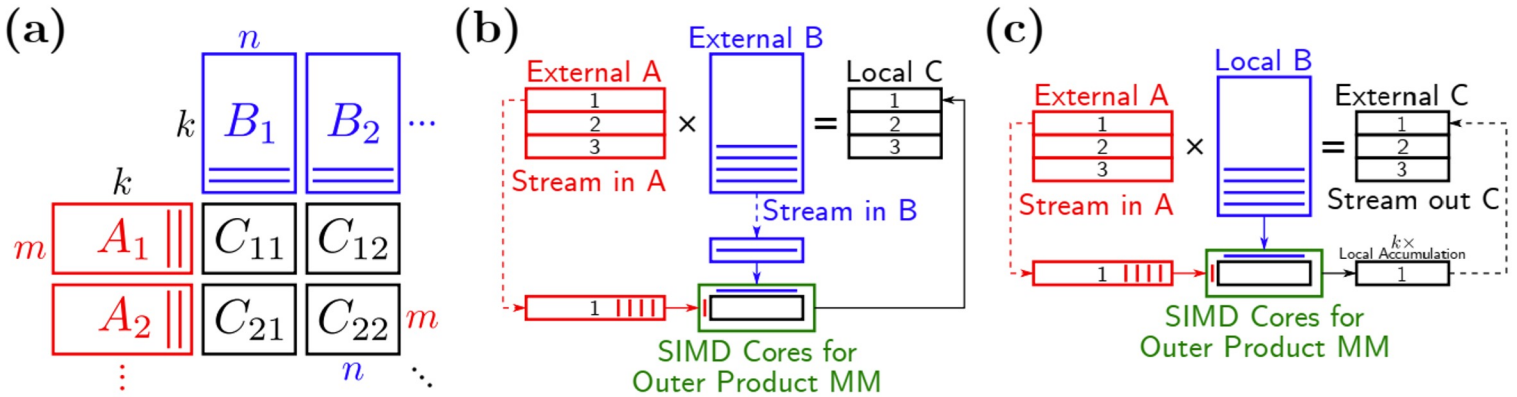
Note: our example is for MM, but this methodology may be applied to other operations with static loop bounds (e.g., tensor contraction)

Tile Sizes and Data Streaming Strategies

We illustrate two general strategies to improve tiled matrix multiplication (a):

(b) Local accumulation to reduce IO requirements
(c) Increasing local input data to increase the amount of computation

We can apply both in balancing IO and computation.



Loop Order Selection

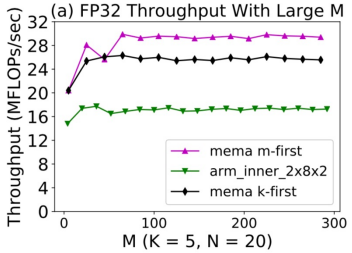
Matrix multiplication can be performed using any order of *M*, *N*, *K*.

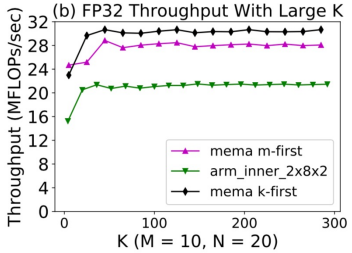
However, depending on the problem sizes (e.g., with very skew matrices), there may be more **reuse opportunities** in specific dimensions.

Algorithm 1: M-first MM using $m \times k \times n$ computation blocks.

```
// loops on M/m × K/k × N/n computation blocks
for t3 = 0 → K, k do
    for t2 = 1 → N, n do
        for t1 = 1 → M, m do
            // loops for a single computation block
            for k* = t3, t3 + k − 1 do
                for j = t2, t2 + n − 1 do
                    for i = t1, t1 + m − 1 do
                        if k* == 0 then
                            C[i][j] ⟵ 0
                        C[i][j] ⟵ C[i][j] + A[i][k*] · B[k*][j];
```

Loop Order IO





We can derive the needed IO for each loop order:

For an N-first schedule which keeps A tiles stationary, our total IO is:

$$IO_{N\text{-first}} = \frac{M}{m} \cdot \frac{K}{k} \cdot \frac{N}{n}(2mn + nk) + \frac{M}{m} \cdot \frac{K}{k}(mk) = MKN\left(\frac{1}{m} + \frac{2}{k}\right) + MK$$

For an M-first schedule which keeps B tiles stationary, our total IO is:

$$IO_{M\text{-first}} = \frac{M}{m} \cdot \frac{K}{k} \cdot \frac{N}{n}(mk + 2mn) + \frac{K}{k} \cdot \frac{N}{n}(kn) = MKN\left(\frac{2}{k} + \frac{1}{n}\right) + KN$$

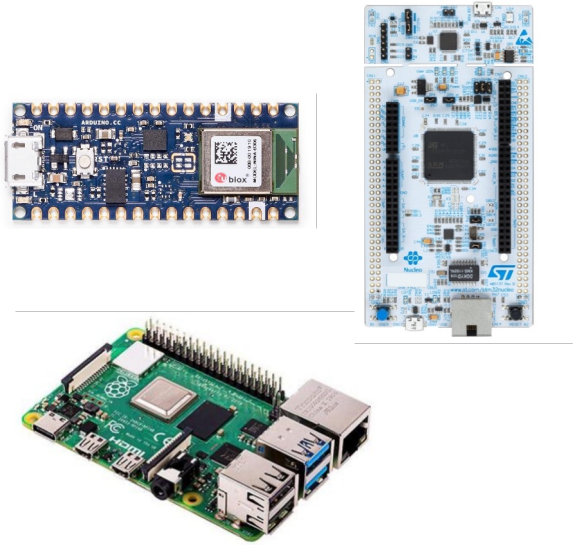
For an K-first schedule which keeps C tiles stationary, our total IO is:

$$IO_{K\text{-first}} = \frac{M}{m} \cdot \frac{K}{k} \cdot \frac{N}{n}(mk + kn) + \frac{M}{m} \cdot \frac{N}{n}(2mn) = MKN\left(\frac{1}{m} + \frac{1}{n}\right) + 2MN$$

Hardware Evaluation

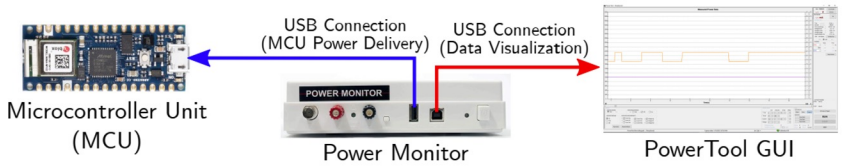
We considered three ARM platforms:

- Arduino Nano 33 BLE (ARMv7E-M Cortex-M4)
- STM32F767ZI Nucleo (ARMv7E-M Cortex-M7)
- Raspberry Pi 4 Model B (ARMv8-A Cortex-A72)



Performance metrics for evaluation:

- Computation throughput (FLOPS)
- External memory IO (bytes)
- Energy consumption (millijoules)



Evaluation Workloads and Comparisons

Evaluation workloads consist of dense MMs for layers of representative neural network models. Matrices and matrix dimensions were obtained from:

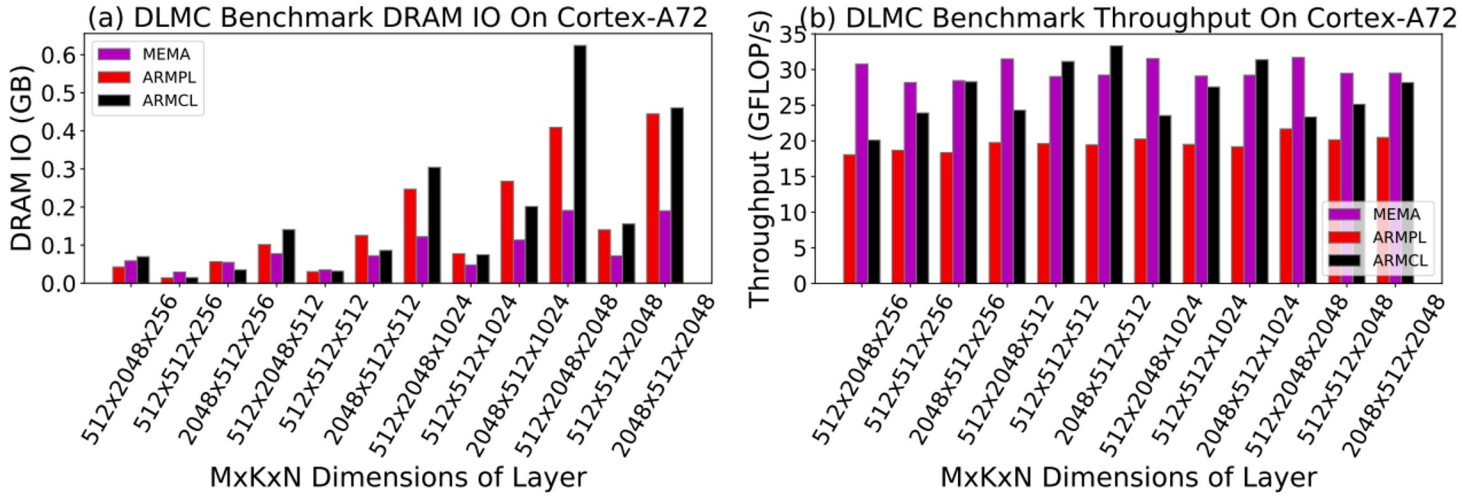
- MLPerf Tiny Inference Benchmark
  - Includes tasks such as visual wake words, image classification, etc.
- Deep Learning Matrix Collection (DLMC)
  - Includes transformer models

We compare to existing libraries:

- Arm Performance Libraries (ARMPL)
  - Standard core math libraries for Arm hardware
- Arm Compute Library (ARMCL)
  - Optimized low-level machine learning functions for Cortex-A
- CMSIS-NN
  - Optimized neural network kernels with support for digital signal processing units

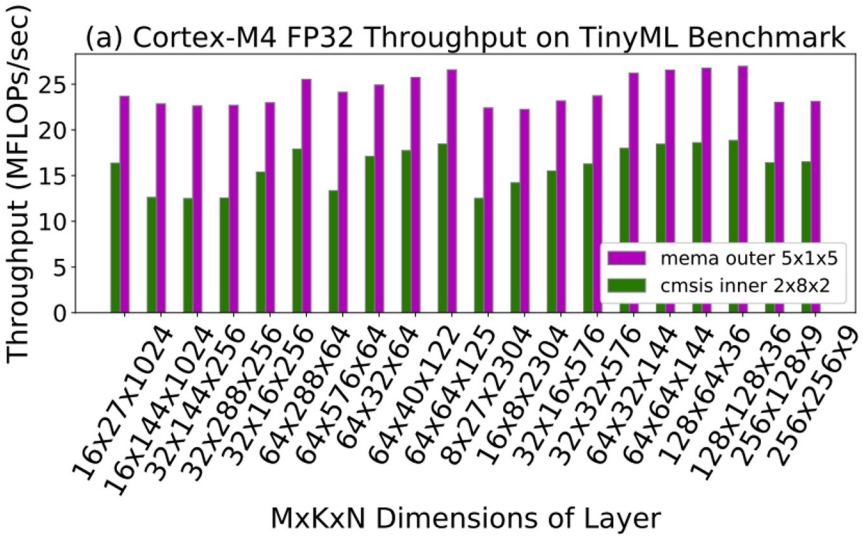
Evaluations: DRAM IO

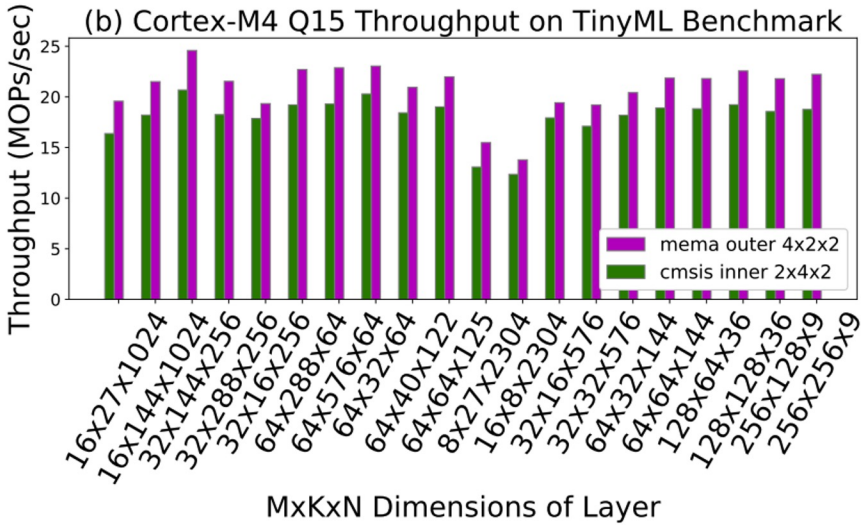
MEMA matches or exceeds the performance of existing libraries while reducing the total IO required



Evaluations: Throughput

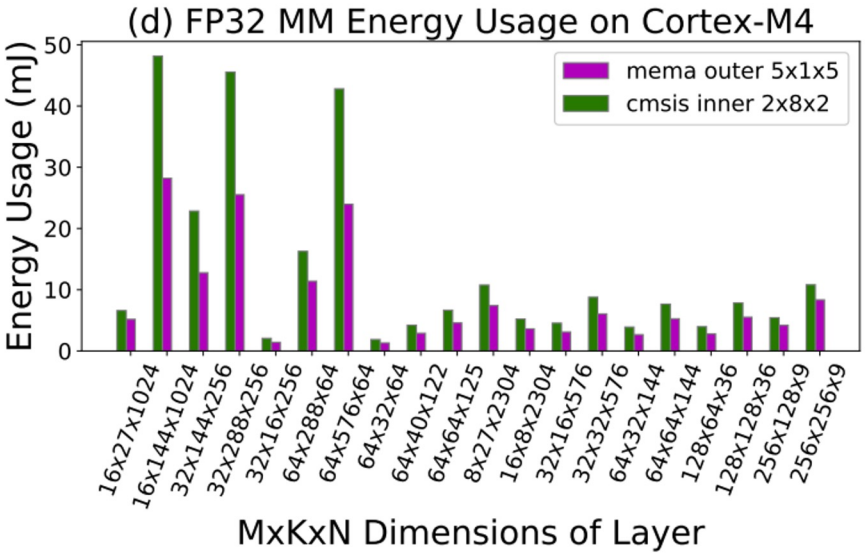
MEMA kernels achieve higher throughput for both FP32 and fixed-point Q15 MM

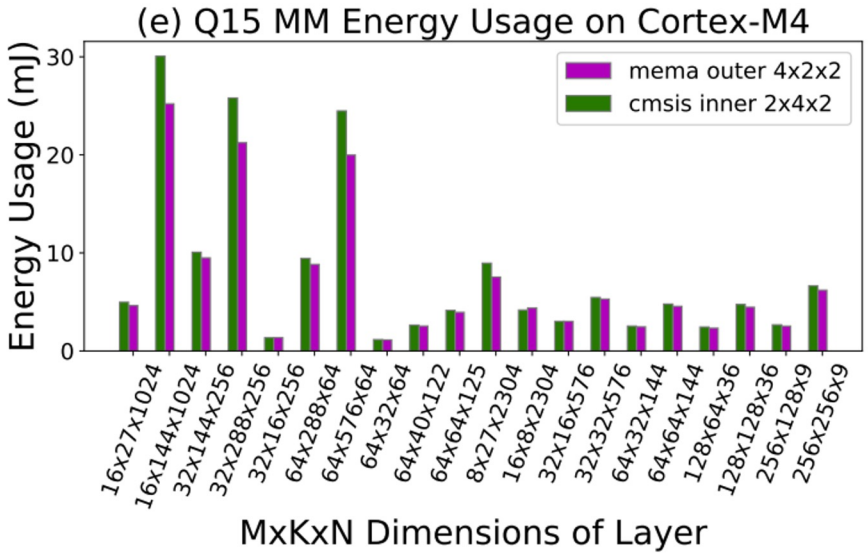




Evaluations: Energy Consumption

Additionally, we observe significant reductions in energy consumption





Conclusion

- The MEMA Runtime Framework enables significant improvements in both **throughput** and **energy consumption** for real-world hardware
- These improvements can enable more powerful applications and longer battery runtimes for devices in the future
- Future work includes integrating automatic polyhedral analysis and additional kernel generation strategies

Copyright Notice

www.tinyml.org