arXiv:2304.05544v1 [cs.LG] 12 Apr 2023

MEMA 运行时框架：最小化微控制器上 TinyML 的外部存储器访问

Andrew Sabot*
asabot
@g.harvard.edu 哈佛大学 美国马萨诸塞州剑桥市

Vikas Natesh*
vnatesh
@g.harvard.edu 哈佛大学 剑桥，马萨诸塞州，美国

H.T. Kung
kung@harvard.edu
哈佛大学 美国马萨诸塞州剑桥市

魏德庭
weiteting
@g.harvard.edu 哈佛大学 美国马萨诸塞州剑桥市

摘要

我们提出了 MEMA框架，用于轻松快速地推导高效的推理运行时，从而最小化 TinyML系统上矩阵乘法的外部存储器访问。该框架在分析确定优化调度和内核时，考虑了硬件资源约束与问题规模，以最小化内存访问。MEMA解决了当前实践中的一个众所周知的问题：最优调度往往只能通过对庞大调度空间进行耗时的启发式搜索来找到。我们将基于 MEMA 推导出的运行时性能与 ARM-based TinyML 系统上的现有最先进库进行了比较。例如，在 ARM Cortex-M4 上针对神经网络基准测试，我们相较于CMSIS-NN 实现了最高 1.8 倍的加速和 44% 的能耗降低。

关键词

TinyML，矩阵乘法，算术强度，外积，内存访问，计算调度，神经网络

1 引言

小型物联网（IoT）设备日益普及，并被应用于包括医疗保健和消费产品在内的越来越多的领域[26]。随着这些设备越来越受欢迎，所收集的数据量不断增加，从而推动了对计算和机器学习（ML）或在这些系统上运行的 TinyML [20], 的需求。然而，由于尺寸和功耗限制，这些设备的可用内存、带宽和计算能力都受到严格限制。此外，TinyML设备可能会作为分布式系统进行部署（例如，用于 AR/VR 工作负载 [12]），并且通信带宽有限。因此，有必要为此类设备支持在各种系统配置下高效执行机器学习应用。

机器学习模型涉及卷积层、全连接层、ReLU、最大池化和批归一化等操作。卷积神经网络（CNNs）的卷积层和全连接层，以及 Transformer 的注意力层中的计算，可以实现为矩阵乘法（MM）。

（例如，参见 [21, 28]）。因此，矩阵乘法占据了相当大一部分运行时间。当矩阵乘法无法完全放入计算硬件的本地存储器（例如中央处理器上的寄存器和缓存）时，就需要与外部存储器进行额外的数据传输。这些额外的输入输出会增加能耗和延迟。

TinyML 硬件的特性和能力可能存在显著差异，因此，对于同一问题，不同设备上的内核和调度的最优选择也可能不同。因此，开发一种能够为种类繁多的 TinyML 系统寻找最优调度的运行时框架，是实现高效推理的关键。

为了刻画系统架构的能力，通常使用屋顶线图（例如，参见 [29]）。斜线的斜率是内存输入输出带宽，水平线是峰值计算吞吐量。拐点的水平位置表示要达到硬件所提供的峰值计算吞吐量所需的最小 算术强度（每次输入输出操作所执行的算术运算次数）。用于最小化内存访问的最优调度是屋顶线图的函数。例如，一些微控制器（微控制器），如 ARMCortex-M4，与其计算能力相比具有相对较高的内存带宽 [15]。对于这些系统，拐点与较小的算术强度相关联。

本文介绍了MEMA 运行时框架，用于为TinyML 生成高效的推理运行时，从而最大限度地减少外部存储器访问。该框架分析硬件和计算问题，以推导出输入输出高效的运行时调度。我们使用Roofline模型来分析MEMA方法，重点关注本地存储器较小的微型设备。对于本文评估的三种架构（ARMCortex-M4、M7 和A72），我们在图2 中展示了其屋顶线图。MEMA 框架通过调度计算来提高算术强度，原因有二：1）当位于拐点左侧时，提高计算吞吐量；2）当位于拐点右侧时，降低所需的输入输出带宽，从而降低内存输入输出的能耗。

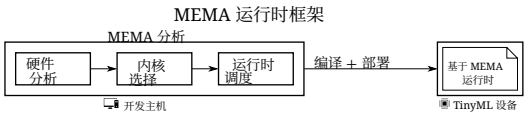


图1：MEMA 运行时框架概述 **ork.**

根据硬件特性选择合适的库和内核，是中央处理器上的常见做法[25]。然而，这些技术尚未应用于微控制器。由于硬件存在差异（例如存储层次结构和可用指令的不同），中央处理器领域与微控制器领域的分析依据大不相同，因此我们无法将中央处理器领域中的决策直接迁移到微控制器领域。因此，本文提出并验证了针对微控制器的基础分析。

第 2 节介绍矩阵乘法的背景知识并概述相关工作。第 3 节概述 MEMA 运行时框架的主要目标，并说明如何通过流式处理这一调度技术，最大限度地提高本地内存中的数据复用

*两位作者对本研究贡献相同。

tinyML 研究研讨会'23，2023年3月，加利福尼亚州圣何塞2023。

，从而使派生出的运行时能够在内存占用受到严格限制的 TinyML 设备上运行。第 4 节详细介绍针对分块矩阵乘法的一般 MEMA 分析流程。第 5 节举例说明如何利用第4节中的分析，在真实硬件上推导出 MEMA 调度。第 6 节介绍我们评估 MEMA 时所采用的方法和基准测试。第 7 节展示了 MEMA 运行时在多种 TinyML 设备上的性能，并与最先进的库进行比较。

本文的贡献如下：

- MEMA 运行时框架可减少生成的运行时的总内存 I/O 需求，并将 I/O 时间隐藏在计算时间中。据我们所知，MEMA 是首个旨在简化 TinyML 系统调度设计难题的此类框架。
- MEMA 将流式处理框架形式化，以支持在内存系统高度受限的微控制器上，对输入和结果分块进行适当的整形和分配（第 4 节）（A、 B 和 C）
- MEMA analytically derives optimal 分块 for multiple bit widths without the need for searching, as shown in Figure 6.
- 展示 MEMA 运行时在真实硬件上的性能的实测结果（第 7 节）。

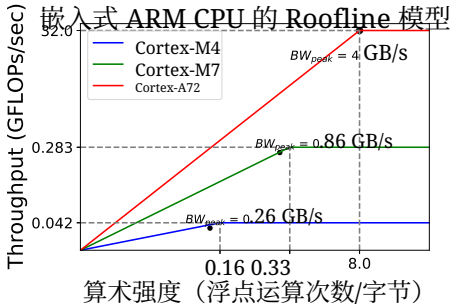


图2：ARMCortex-M4、M7和 A72的屋顶线图。每个设备的拐点均以黑点表示。拐点越靠右的系统越受带宽限制，这表明更高的算术强度将提高吞吐量。相反，拐点越靠左的系统，相对于计算吞吐量而言具有更大的内存带宽。

2 背景与相关工作

2.1 将矩阵乘法划分为计算块

对于无法将 A、 B 和 C 全部放入本地内存的矩阵乘法，我们必须将其划分为更小的 $m \times k \times n$ 计算块，以确保每个计算块的输入都能放入本地内存（图3a）。

算法1描述了一种分块矩阵乘法，其中对于最外层循环中的每个 t_3 值，都会对两个子矩阵（$M \times k$ 和 $k \times N$ 矩阵）执行基于外积的矩阵乘法。我们将该方案记作 $M \to N \to K$，以反映嵌套循环的顺序。对于这种外积矩阵乘法，该方案执行 M 优先的块计算（t_1 位于最内层循环中），而不是 N 优先的块计算。共有**六种方案**用于调度

算法1： M优先的矩阵乘法，使用 $m \times k \times n$ 计算块。

// 对 $M/m \times K/k \times N/n$ 计算块进行循环
对 $t_3 = 0 \to K$， $t_3 += k$ **执行**
  对 $t_2 = 0 \to N$， $t_2 += n$ **执行**
    对 $t_1 = 0 \to M$， $t_1 += m$ **执行**
      // 单个计算块的循环
      对于 $k^* = t_3 \to t_3 + k$， $k^{*}{+}{+}$ **执行**
        对于 $j = t_2 \to t_2 + n$， $j{+}{+}$ **执行**
          对于 $i = t_1 \to t_1 + m$， $i{+}{+}$ **执行**
            $C[i+t_1 \cdot m][j+t_2 \cdot n] \longleftarrow C[i+t_1 \cdot m][j+t_2 \cdot n] + A[i+t_1 \cdot m][k^*+t_3 \cdot k] \cdot B[k^*+t_3 \cdot k][j+t_2 \cdot n]$;

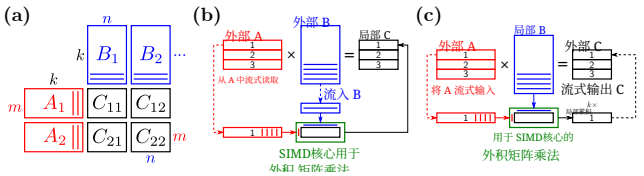


图3：（a）算法1中，使用 $m \times k \times n$ 个计算块对 t_3 在$M \times k$和 $k \times N$矩阵的每个取值进行矩阵乘法。 A_i和 B_i 分别是 A 和 B的分块，线条分别表示列向量和行向量。 C_{ij} 是 C的分块。（b）我们可以将 C的各行保留在本地内存中，并流式读入 A 和 B的各行。（c）通过将计算出的 C 各行流式写入外部存储器，我们可以让输入输出时间与计算时间重叠。当 k 足够大时，计算 C 每一行所需的时间超过将该行写回外部存储器的时间。与（b）相比，该方案只需将 C 的一行作为中间结果保留在内存中。（b）和（c）都是 MEMA 可生成的调度示例。

计算块： $M \to N \to K$、 $N \to M \to K$、 $M \to K \to N$、 $N \to K \to M$ $K \to M \to N$ $K \to N \to M$ 、 和 。

2.2 静态驻留与流式数据

每次矩阵乘法计算都必须从外部存储器加载 A、 B和 C 分块。当数据在本地内存中静态驻留时，我们能够减少从外部存储器获取的分块数量。我们将为多次计算保留在本地内存中的数据称为静态驻留数据（例如图3b中的 B 矩阵）。从外部存储器获取并在被逐出前仅使用一次的数据（例如图3b中的 A 矩阵）则称为“流式数据”。累积的结果可以流式写入外部存储器（例如图3b中计算得到的 C 各行）。以往的研究（例如 [14], ）已使用流式处理技术，但我们的创新在于自动化运行时调度设计时使用流式处理。

借助 MEMA，我们可以减少与外部存储器之间流式传输的数据总量。当计算时间长于流式传输数据项所需的时间时，我们便能隐藏流式传输耗时（图3），并减小所需的本地存储器大小。将 C 的一部分写回所需的输入输出操作，可以与计算 C 下一部分的过程重叠。我们假设带宽足以将矩阵 A 也从外部存储器流式传入（Cortex-M4 就是如此，其拐点在图 2c 中更靠左）。当固定不动的是 A、 B 或 C 时，流式处理方法也类似。对于这三种方法，较小的

分块 C 会在原位累加，以减少数据移动。如果外部带宽不足，可以对流式传入的数据采用压缩技术，例如 [31] 。

2.3 相关工作

CMSIS-NN [18] 是一组用于常见神经网络运算的内核，专注于提升吞吐量和延迟方面的性能，并尽量减少神经网络在 ARM Cortex-M 微控制器上的内存占用。由于采用内积并固定循环顺序，CMSIS-NN 无法最大限度地实现数据复用。

除了 CMSIS-NN 之外，近期的框架（例如 MCUNet[20] ）还会针对卷积等神经网络层运算执行循环分块和循环展开。MCUNet 在选择分块大小时仅考虑问题规模和可用的本地内存，并未明确减少内存带宽使用。相比之下，我们的工作同时利用本地内存和特定设备的 Roofline 特征来推导高效的分块大小。此外，我们利用外积和循环重排来增加数据复用，从而降低延迟和能耗（图 6）。

ARM 为 ARM Cortex-A 设备上的矩阵乘法提供了两个库：用于机器学习的 ARM Compute Library（ARMCL [6]）和 ARM Performance Library（ARMPL [5]）。虽然 ARMPL 和 ARMCL 在矩阵乘法中采用基于外积的方法，但其计算调度和分块策略基于 Goto 算法 [13]，该算法不会减少输入输出带宽的使用。通过减少外部访问，我们能够超越 ARMCL 和 ARMPL（见图 7）。Goto 算法 [13] 是一种基于数据流式处理的经典算法，用于在中央处理器上实现高性能矩阵乘法，是 OpenBLAS [30] 和 Intel MKL [2]的基础。CUTLASS [4] 是一个面向 Nvidia GPU 的开源 C++ CUDA BLAS 库。CUTLASS 与我们的工作相似之处在于，二者都在矩阵乘法中使用外积形式。

TensorFlow Lite Micro (TFLM) [10] 是面向嵌入式系统深度学习的机器学习推理框架。硬件厂商可以为 TFLM 贡献自己的内核，使程序员能够将机器学习模型部署到多种架构上。目前，TFLM 在对嵌入式硬件平台进行基准测试时会使用 CMSIS-NN 库 [7]。本文展示了，与 TFLM 的 CMSIS-NN 内核相比，我们的内核能够提升能耗效率、带宽利用率和计算吞吐量。

3 MEMA 框架

MEMA运行时框架结合硬件特性、问题规模信息（例如矩阵乘法的输入矩阵大小）、输入输出分析和调度，生成高效的推理运行时。本节将介绍MEMA框架的目标，并说明MEMA如何利用流式处理最大限度地实现数据复用，以及在对无法放入本地内存的输入进行矩阵乘法计算分区时，如何重叠计算与输入输出时间。

TinyML设备受外形尺寸和有限能耗预算的限制，其内存容量和速度均受到约束。MEMA框架通过以下目标应对这些挑战：减少外部存储器访问次数，并通过流式处理将输入输出时间隐藏在计算时间内，从而高效利用可用的本地内存。通过减少内存访问，该框架可能降低机器学习推理任务的能耗 TinyML 设备。这使 MEMA 能够让输入输出与计算重叠，并推导出不受输入输出瓶颈限制的运行时。

3.1 MEMA框架概述 k

给定一个矩阵乘法 $C = C + A \times B$，其中 A 为 $M \times K$，且 B 为 $K \times N$，MEMA框架可以在本地存储器大小的限制下，自动推导出能够最大限度减少外部存储器访问的最优调度，并采用以下技术最大限度地实现数据复用：（1）分块形状设计；（2）矩阵操作数流式传输；（3）循环顺序选择。

该框架首先利用目标硬件的信息（例如寄存器数量、本地存储器大小和外部存储器带宽）来确定最佳分块大小（第 4.1 节）。系统会自动推导分块大小，以最大化每个分块乘法的算术强度（旨在提高计算吞吐量并降低输入输出的能耗）。根据分块大小，MEMA 可以从现有库中选择适用于目标平台的内核，或生成经过优化的外积内核。随后，综合考虑可能的分块大小和矩阵乘法问题的规模，选择能够最大限度减少外部输入/输出（第4.4节）并降低所需输入/输出带宽的调度。接着，将这些内核和调度编译为 MEMA 运行时，并部署到 TinyML 设备上，如图1所示。

4 MEMA运行时推导分析

本节介绍针对两种硬件平台上的矩阵乘法（MM）进行的 MEMA 分析：单核微控制器和多核物联网设备。我们的分析仅限于矩阵乘法操作，但该框架可扩展至其他具有静态循环边界的多重循环归约操作，例如直接卷积和张量缩并。MEMA 分析首先通过解析推导出适用于给定硬件、可最大化算术强度的分块大小。然后，利用推导出的分块方式，我们选择一种循环顺序，以最大化连续分块计算之间的数据复用。我们表明，与流式处理相关的外部存储器输入输出量由分块大小和矩阵乘法问题规模共同决定。同时，驻留数据的输入输出量仅由矩阵乘法问题规模决定。MEMA 选择流式处理和驻留数据的输入输出总量最低的循环顺序。

4.1 推导微控制器的分块大小

考虑图3a所示的外积：将一个 $m \times 1$维向量 A 与一个 $1 \times n$维向量 B 相乘，生成一个驻留于本地的 $m \times n$ 分块 C。为简化讨论，我们只计算乘法次数，不包括加法。为最大限度地提高数据复用率，我们最大化计算量（mn次乘法）与输入输出量（$m+n$个输入值）的比值，即算术强度 $\frac{mn}{m+n}$。选择正方形分块 $m = n = t$时，该比值达到最大。假设有一款微控制器，可用作数据复用本地内存的寄存器共36个。所有外积输入（$m + n + mn$个值）都必须放入本地内存，即 $2t + t^2 \leq 36$。对 t求解可得，该微控制器的最优分块大小 C为 5×5。

在此示例中选定的 C 分块将在使用图3c方案时充当中间本地存储，以支持输入流式处理与计算。虽然我们在此通过简单的算术强度论证推导分块尺寸，但也可以将其他分块技术与 [8, 16] MEMA配合使用。

4.2 为多核物联网设备推导分块大小

与微控制器不同，物联网设备（例如 [1, 11, 24] ）包含多个低功耗 RISC-V 或 ARM 内核，以及多级内存。由于我们必须在多个内存级别进行分块，因此可能的调度数量要多得多。例如，对于具有3个内存级别的设备，我们在每个级别都有 6种矩阵乘法循环顺序可供选择，总共得到 $6^3 = 216$ 种可能的循环顺序。此外，每个内存级别的分块都必须根据可用的本地内存、带宽和内核数量进行适当设置。

不同于 [9, 19],中为获得最优分块而搜索庞大调度空间的方法，我们使用 CAKE [17],——一种多核矩阵乘法分块与调度算法。该算法在计算划分和分块调度中采用**恒定带宽（CB）分块**。CB 分块是指这样一种计算分块：当计算在本地内存中进行时，即使使用额外的核心，所需的片外带宽仍保持不变。CAKE 根据可用的片外 DRAM 内存带宽、核心数量和可用本地内存，通过调整 CB 分块的形状（即纵横比）和大小来控制其算术强度。使用 CAKE 分块，我们可以提高可用计算能力的利用率，而无需相应增加片外内存带宽。增加核心数量时，CAKE 确实需要更多本地内存，但与片外带宽相比，本地存储器大小通常足够。

假设我们希望将核心数量增加 p倍。CAKE 将计算块从 $m \times k \times n$ （如图3所示）重塑为形状 $pm \times k \times pn$。CAKE 还会将较大的C 分块（尺寸为 $pm \times pn$）固定在片上内存中，同时分别流式载入 A和 B的较小 $pm \times k$和 $k \times pn$分块。设 $p \cdot f$ 为系统的峰值 FLOPs，其中 f为单核峰值。为了在本地内存中计算 CB块，CAKE 在时间 $T = \frac{pm \cdot k \cdot pn}{p \cdot f}$ 内执行 $pm \cdot k \cdot pn$ 次乘累加操作，并使用 $IO = pmk + pnk$次片外存储器访问。所需的片外带宽因此为：

$$BW_{off-chip} = \frac{IO}{T} = \frac{p \cdot k \cdot (m+n)}{pm \cdot k \cdot pn} \cdot p \cdot f = \frac{m+n}{mn} \cdot f$$

请注意，由于 p 因子相互抵消，CAKE 的带宽使用量与核心数量无关。鉴于CAKE 的CB分块形状，我们可以通过选择 m、k 和 n，使得 A、B和 C 个分块能够装入本地内存（LM）中，即 $pmk + kpn + p^2mn \leq LM$。这里，$LM$ 是由所有核心共享的片上内存，每个核心一次计算一个 $m \times k \times n$ 子分块。在不失一般性的情况下，我们可以继续根据每个核心可用的寄存器或私有本地内存，对 $m \times k \times n$ 子分块进行分块（参见第 7 节）。

4.3 块间计算的循环顺序

采用不同的循环顺序计算计算块，可能会导致同一矩阵乘法产生不同总量的外部存储器访问。CAKE 不会通过重排循环来尽量减少外部存储器访问，而是仅采用以 K 优先的块调度（将 K 维作为内层循环，使部分结果保持驻留）。由于循环顺序决定了流式访问模式，因此输入输出总量会有所不同。对于非均衡矩阵形状，部分结果驻留调度可能无法尽量减少片外存储器访问。MEMA 通过选择能最大限度减少外部存储器访问总量的循环顺序加以改进，即使该顺序无法使部分结果保持驻留。

对于 $M \times K$ 矩阵 A 与 $K \times N$ 矩阵 B 之间的分块矩阵乘法（见第 2.1 节），给定循环顺序下的总输入输出量可计算为流式处理输入输出量与驻留输入输出量之和：

$$(\#\text{ of blocks}) \cdot (IO_{streaming} \text{ per block}) + IO_{stationary}$$

对于以 N 优先的调度（ A 分块驻留），总输入输出量为：

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

对于以 M优先的调度（ B 分块驻留），总输入输出量为：

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

对于以 K 优先的调度（ C 分块驻留），总输入输出量为：

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

$2mn$ 中的系数 2 表示读取并 写入部分 C 分块。

4.4 根据分块尺寸和输入矩阵大小选择循环顺序

使用第 4.3 节中的 I/O 方程，MEMA 会选择使 I/O 最小化的调度。分块大小决定流式处理的内存访问次数。A 和 B 的大小决定驻留数据的内存访问次数。可以通过计算第 2.1 节所列 6 种可能循环顺序各自的外部存储器访问次数，来选择使外部存储器访问次数最少的循环顺序。本节接下来将举两个矩阵乘法循环顺序选择的示例。

考虑使用非正方形分块（$n > k = m$）计算 $M = K = N$。B 和 C 分块更大，因此流式处理时需要更多输入输出。在以 N 优先的循环顺序中，A 分块保持不动，同时流式处理 B 和 C 分块。但是，我们可以通过采用以 M 优先或以 K 优先的顺序，并流式处理 A 分块来减少输入输出。在此示例中，以 M 优先和以 K 优先的顺序等效，因为其他所有维度都相等。

现在考虑 A 和 B 之间的矩阵乘法（MM），其中 $M > N$ 和 $M > K$。假设使用正方形矩阵块（$n = m = k$）计算该矩阵乘法。使用 N-优先调度计算该矩阵乘法所需的外部存储器访问总量，比使用 M-优先调度更多。相比之下，M-优先顺序的输入输出较少，因为 B 分块可以重复利用 $\frac{M}{m}$ 次，这比其他方向更多，因为 $M > N$ 和 $M > K$。当分块大小相等时，输入输出函数中的流式处理项与循环顺序无关。

5 MEMA 针对真实硬件的推导

5.1 评估硬件特性

在本节及后续各节（第6 节和第 7 节）中，我们在多个代表流行嵌入式平台的微控制器上，将 MEMA 的性能与用于矩阵乘法的最先进的库进行比较评估：Arduino Nano 33 BLE（ARMv7E-M Cortex-M4）、STM32F767ZI Nucleo（ARMv7E-M Cortex-M7）以及 Raspberry Pi 4 Model B（ARMv8-ACortex-A72）。MEMA 在各自平台上使用 C++ 和编译器内置函数实现（开源地址：https://github.

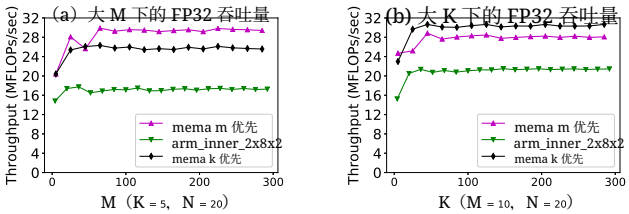


图 4：Cortex-M4 上MEMA 在各种矩阵乘法问题中的性能。通过调度以最小化输入输出，MEMA最大化了形状偏斜的矩阵的吞吐量。(a) M-first 调度优于 K-first 调度，当$K = 5$时，正如 MEMA 的分析（第 5.2节）所预测的那样。同样，(b) 显示 K-first 调度优于 M-first调度，对于$K > 10$而言。MEMA 调度优于 ARM 库（arm_inner_2x8x2）。

com/vnatesh/MEMA-MM)。Cortex-M4 和 Cortex-M7 的本地内存非常有限：只有 32 个浮点寄存器，以及 16 个用于 DSP 加速定点运算的寄存器。Cortex-M4 配备 256KB 外部存储器（静态随机存取存储器）和 1MB 闪存，用于存储额外的数据和代码。Cortex-M7 的静态随机存取存储器和闪存容量约为 M4 的两倍。与此同时，Cortex-A72 的计算和内存资源多于 Cortex-M4 和 M7，但相对于其计算能力，仍受限于 DRAM 带宽、本地存储器大小和本地存储器带宽（见图 2）。

5.2 在 Cortex-M4 上推导的 MEMA 调度

在本节中，给定 $M, K,$ N ，我们将 MEMA 的分析应用于 ARM Cortex-M4，以最小化矩阵乘法问题中的内存访问。在根据设备的本地存储器大小和层次结构（第 4.1 节）确定最优分块大小后，我们推导出使用六种计算块调度中任意一种的条件（第 4.3 节）。例如，考虑在 M 优先和 K 优先调度之间的选择。为了最小化输入输出，我们只应在 IO_M 优先调度的 $\leq IO_K$ 优先且 IO_M 优先的 $\leq IO_N$ 优先时使用 M 优先调度。使用第 4.3 节中的方程，我们得到以下不等式，作为选择 M 优先调度的条件：

$$MKN\left(\frac{2}{k}+\frac{1}{n}\right)+KN \leq MKN\left(\frac{1}{m}+\frac{1}{n}\right)+2MN$$

$$MKN\left(\frac{2}{k}+\frac{1}{n}\right)+KN \leq MKN\left(\frac{2}{k}+\frac{1}{m}\right)+MK$$

$$\implies K \leq \left(\frac{2M}{1+M(\frac{2}{k}-\frac{1}{m})}\right),\ N \leq \left(\frac{M}{1+M(\frac{1}{n}-\frac{1}{m})}\right)$$

由于 Cortex-M4 有 36 个寄存器可用于本地数据复用，假设我们使用第 4.1 节推导出的、用于驻留数据的正方形 5×5 分块，即 $m = k = n = 5$。那么，上述分析表明，当 K 大致为≤ 10 且 $M \geq N$ 时，我们应采用 M-优先调度，因为此时上述两个不等式成立。我们通过图 4a 中的实测结果确认这一选择：当 $K = 5$时，在 Cortex-M4 上， M-优先调度的表现优于 K-优先调度。同样，图 4b 显示，当 K超过 10，且 M和 N固定时， K-优先调度的表现优于 M-优先调度。我们可以通过比较 IO_N-优先和 $\leq IO_K$-优先调度，在 N和 K-优先调度之间作出选择，从而得到与上述类似的不等式。



图 5：用于采集通过 USB 连接的微控制器数据的功率监测器设置。监测器通过 PowerTool 图形用户界面实时报告电压和电流 [22]。

6 神经网络基准测试与能耗测量方法

神经网络基准测试。我们测量各种基准测试中的神经网络模型各层权重矩阵与数据矩阵进行矩阵乘法时的计算吞吐量（FLOPs/秒）、外部存储器输入输出量（字节）和能耗（焦耳）。对于 Cortex-M4 和 M7，我们使用 MLPerf Tiny 基准测试 [7], , 其中包含用于关键词识别、视觉唤醒词和图像识别等任务的模型。我们从该基准测试中的矩阵提取各层权重矩阵和数据矩阵的维度，并应用 *im2col* [27]。对于 ARM Cortex-A72，我们使用深度学习矩阵集合（DMLC） [23]提供的 Transformer 模型矩阵。对于所评估的矩阵规模，打包开销较大，因此我们在 M4 和 M7 上的内核无需打包。

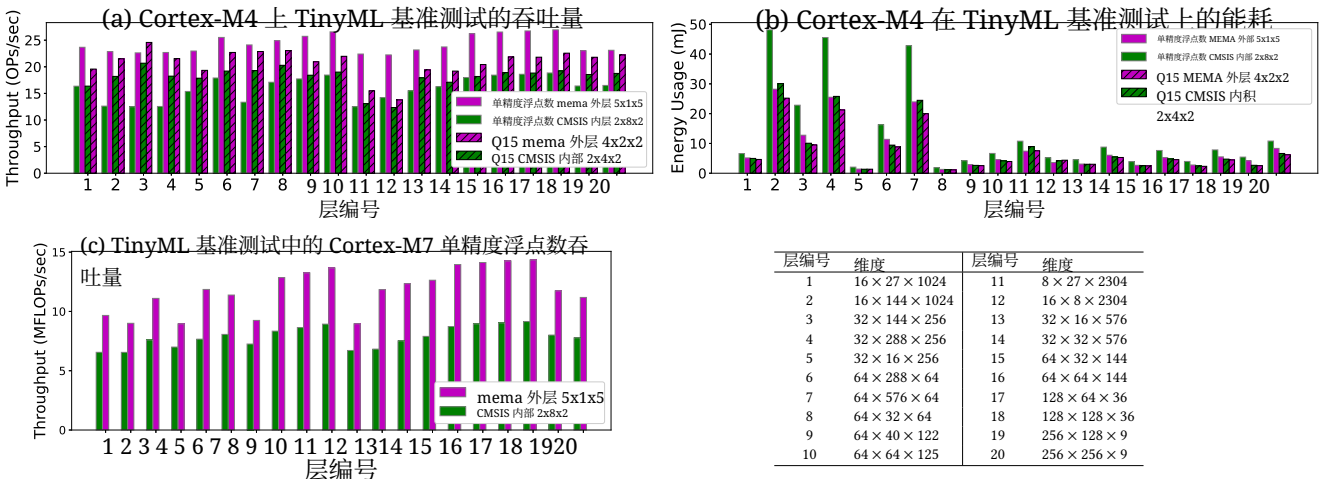
能耗测量设置。 为测量能耗，我们遵循 MLPerf Tiny 基准测试指南 [7]。我们使用 Monsoon Solutions 低压功率监测器 [22], , 其电流分辨率为 $50\,\mu$A，电压分辨率为 $125\,\mu$V。功率监测器每隔 $200\,\mu$s 采样一次电压和电流，并通过主机台式电脑上的 PowerTool 图形用户界面报告数据。微控制器连接到功率监测器的 USB 通道，并启用 USB 直通功能，以便与主机台式电脑上的 PowerTool 图形用户界面通信（图 5）。微控制器以 5V 运行，处于 USB 通道 2.1 V 至 5.4 V 的电压范围内。微控制器的电流消耗在 16 至 22 mA 之间，误差范围为 ~2%。我们报告 Arduino Nano 33 BLE（ARMv7E-M Cortex-M4）的能耗，因为它能代表其他设备。矩阵乘法的能耗测量结果取数百次试验的平均值。

7 结果与评估

7.1 Cortex-A72 上的 DLMC 基准测试

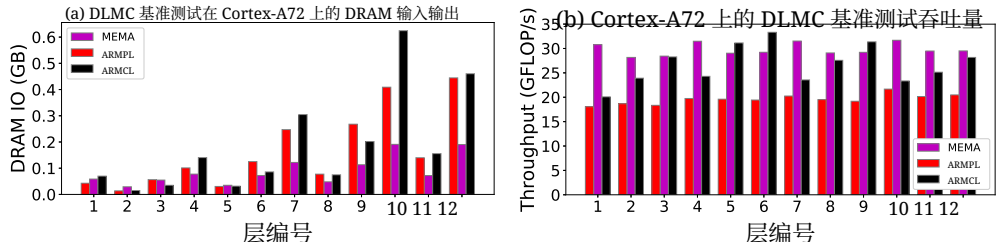
我们使用 MEMA、ARMCL 和 ARMPL，在 ARM Cortex-A72 CPU 上对 DLMC 基准测试中的 Transformer 模型矩阵执行矩阵乘法，并使用 *perf* [3]收集性能数据。我们通过 ARM PMU 事件计数器（来自 DRAM 的 L2 缓存填充）监控 DRAM 访问。Cortex-A72 包含四个核心（$p = 4$）、一个共享的 L2 缓存，以及每个核心的私有 L1 缓存。如第 4.2 节所示，我们将矩阵乘法划分为 CB 分块，每个核心一次计算一个 $m \times k \times n$ 子块，并使用 $m = k = n$和 $pmk + kpn + p^2mn \leq L2$。我们进一步将 $m \times k \times n$ 子块划分为单独的 $8 \times k \times 12$ 外积，这些外积可利用每个核心可用的 128 个 32位寄存器进行计算。 $8 \times k \times 12$形状是将第 4.1 节的微控制器分析调整为适用于 CPU 单核后确定的。

ARMCL 在某些矩阵乘法问题上达到峰值吞吐量（图 7b），但可能无法高效调度。ARMPL 和 ARMCL 根据经典的 Goto 算法对矩阵乘法计算进行分块 [13], , 这需要更多 DRAM 带宽。相比之下，MEMA 达到



层编号	维度	层编号	维度
1	$16 \times 27 \times 1024$	11	$8 \times 27 \times 2304$
2	$16 \times 144 \times 1024$	12	$16 \times 8 \times 2304$
3	$32 \times 144 \times 256$	13	$32 \times 16 \times 576$
4	$32 \times 288 \times 256$	14	$32 \times 32 \times 576$
5	$32 \times 16 \times 256$	15	$64 \times 32 \times 144$
6	$64 \times 288 \times 64$	16	$64 \times 64 \times 144$
7	$64 \times 576 \times 64$	17	$128 \times 64 \times 36$
8	$64 \times 32 \times 64$	18	$128 \times 128 \times 36$
9	$64 \times 40 \times 122$	19	$256 \times 128 \times 9$
10	$64 \times 64 \times 125$	20	$256 \times 256 \times 9$

图6：MEMA 和 ARM CMSIS-NN 在ARM Cortex-M4和M7上使用TinyML基准测试中的矩阵执行 MM时的吞吐量和能耗。每个柱状组代表某一层的 MM。每层的输入矩阵维度显示在右下方的表格中。对于单精度浮点数 MM，MEMA 的吞吐量最高可达 CMSIS-NN 的 1.8 倍，能耗降低44%。对于微控制器数字信号处理器上的Q15 算术运算，仅有 12 个寄存器可用，从而限制了分块选项。对于 Q15MM，MEMA的吞吐量最高可达 CMSIS-DSP 的 1.2 倍，能耗降低 20%。



层编号	维度 ($M \times K \times N$)
1	$512 \times 2048 \times 256$
2	$512 \times 512 \times 256$
3	$2048 \times 512 \times 256$
4	$512 \times 2048 \times 512$
5	$512 \times 512 \times 512$
6	$2048 \times 512 \times 512$
7	$512 \times 2048 \times 1024$
8	$512 \times 512 \times 1024$
9	$2048 \times 512 \times 1024$
10	$512 \times 2048 \times 2048$
11	$512 \times 512 \times 2048$
12	$2048 \times 512 \times 2048$

图 7：在 ARM Cortex-A72 上对 DLMC 基准测试中的 Transformer 模型矩阵进行相乘时，MEMA、ARMPL 和 ARMCL 的吞吐量和外部存储器输入输出 [23]。右侧表格显示了每一层的输入矩阵维度。通过最小化外部存储器访问 (a)，MEMA 通常能够实现峰值计算吞吐量 (b)。

通过运行时调度和本地内存中的 CAKE 分块，最大限度地减少输入输出（图7a），从而为所有输入形状实现高吞吐量。

7.2 ARMCortex-M4 和 M7 上的 MLPerf Tiny 基准测试

当在Cortex-M4和 M7上对MLPerf Tiny 基准测试 [7]中的矩阵执行矩阵乘法时，我们将 MEMA与ARMCMSIS-DSP 进行了比较。CMSIS使用内积内核对矩阵乘法计算进行分块，例如基于通用矩阵乘法的卷积。比如，单精度浮点数矩阵乘法计算被分块为 A 的 2×8行分块与 B 的 8×2 列分块之间的内积，从而生成 C 的 2×2 分块（图6中的 cmsis inner2x8x2）。相比之下，MEMA通过 A 的 5×1 列向量与 B 的 1×5行向量之间的外积对矩阵乘法进行分块，从而生成保存在寄存器中的 C 的 5×5 分块（图 6中的 mema outer 5x1x5）。MEMA的性能最高可超过 ARM 内核 1.8×，同时将能耗降低最高达 44%。

对于 Cortex-M4/7 数字信号处理器上的饱和定点 16 位算术（Q15），较小的本地内存（十二个 32位寄存器）将内核选择限制为少数几种形状。此外，输入矩阵中的数值对以 32位字的形式存储，因此在进行双乘累加 SIMD 计算之前，必须先解包每对 16 位数值。这种位打包会使计算停滞数个周期，导致屋顶线脊点远在左侧。尽管 Q15 计算严重受计算限制，MEMA 仍通过减少外部输入/输出，提升了相较于 CMSIS 的吞吐量。

CMSIS Q15 矩阵乘法计算被分块为 2×4 个 A 的分块与 4×2 个 B 的分块之间的内积，以生成一个 2×2 个 C 的分块（图 6 中的 cmsis inner 2x4x2 ）。MEMA 将 4×2 个 A 的分块与 2×2 个 B 的分块之间的矩阵乘法分块，以生成 4×2 个驻留的 C 分块（图 6 中的 mema outer 4x2x2）。MEMA 和 CMSIS 使用相同的 SIMD 指令，每个周期执行多次乘累加操作，但 MEMA 的分块调度减少了必须从外部存储器流式传输 B 的次数（MEMA 为 $\frac{M}{4}$ 次，而 CMSIS 为 $\frac{M}{2}$ 次），如第 4.4 节的分析所示。因此，MEMA 的计算吞吐量比 CMSIS 高 20%，同时能耗最多降低 23%。

8 结论

我们提出了 MEMA框架，用于生成推理运行时，从而最大限度地减少微控制器（微控制器）上 TinyML 的外部存储器访问。通过减少外部存储器访问，MEMA 生成的调度可以提高计算吞吐量并降低能耗。

MEMA 可通过选择适合硬件的数据流式处理、分块形状设计和循环顺序方案，适配具有不同屋顶线脊点的各种 TinyML 微控制器（图3b 和 c）。例如，对于 Cortex-M4 上的神经网络基准测试，MEMA 生成的运行时调度比最先进的 CMSIS-NN 库快 1.8 倍，能耗降低 44%（图6）。本研究表明，MEMA 等简单的 TinyML 微控制器运行时框架能够显著减少内存访问和能耗。

9 致谢

本研究部分由美国空军研究实验室资助，资助编号为 FA8750-18-1-0112 和 FA8750-22-1-0500；另获 Meta Platforms Technologies 资助，资助编号为 A51540。

参考文献

[1] [无日期]。ARM® Cortex®-A72 MPCore 处理器技术参考手册。https://developer.arm.com/documentation/100095/0003/?lang=en [2] [无日期]。*Intel oneAPI* 数学内核库。[3] [无日期]。Perf Wiki。https://perf.wiki.kernel.org/index.php/Main_Page。[4] 2020。CUTLASS：CUDA C 中的快速线性代数++。https://developer.nvidia. com/blog/cutlass-linear-algebra-cuda/。[5] Arm Limited 2021。*Arm* 性能库参考指南。Arm Limited。https://developer.arm.com/documentation/101004/latest/ [6] Arm Limited 2022。*Arm* 计算库参考指南。Arm Limited。https: // arm-software.github.io/ComputeLibrary/latest/ [7] Colby Banbury、Vijay Janapa Reddi、Peter Torelli、Jeremy Holleman、Nat Jeffries、Csaba Kiraly、Pietro Montino、David Kanter、Sebastian Ahmed、Danilo Pau、Urmish Thakker、Antonio Torrini、Peter Warden、Jay Cordaro、Giuseppe Di Guglielmo、Javier Duarte、Stephen Gibellini、Videet Parekh、Honson Tran、Nhan Tran、Niu Wenxu 和 Xu Xuesong。2021。MLPerf Tiny 基准测试。*arXiv*：*2106 07597 [ ]* 2021 2106 07597 [ ]。*cs*（8月）。arXiv：。cs [8] Uday Bondhugula、Muthu Baskaran、Sriram Krishnamoorthy、J. Ramanujam、Atanas Rountev 和 P. Sadayappan。2008。多面体模型中的通信最小化并行化与局部性优化自动变换。载于编译器构造（计算机科学讲义），Laurie Hendren（编）。Springer，柏林、海德堡，132–146。https://doi.org/10.1007/978- 3-540-78791-4 9_[9] Alessio Burrello、Angelo Garofalo、Nazareno Bruschi、Giuseppe Tagliavini、Davide Rossi 和 Francesco Conti。2021。DORY：在低成本物联网微控制器上自动端到端部署真实世界的深度神经网络。*IEEE*计算机学报70，8（2021），1253–1268。https://doi.org/10.1109/TC.2021.3066883[10] Robert David、Jared Duke、Advait Jain、Vijay Janapa Reddi、Nat Jeffries、Jian Li、Nick Kreeger、Ian Nappier、Meghna Natraj、Tiezhen Wang、Pete Warden 和 Rocky Rhodes。2021。TensorFlow Lite Micro：面向 TinyML 系统的嵌入式机器学习。机器学习与系统论文集3（2021年3月），800–811。[11] Eric Flamand、Davide Rossi、Francesco Conti、Igor Loi、Antonio Pullini、Florent Rotenberg 和 Luca Benini。2018。GAP-8：面向物联网边缘人工智能的 RISC-V 片上系统。载于*2018*年*IEEE*第*29*届应用专用系统、架构与处理器国际会议（*ASAP*）。1–4。https://doi.org/10.1109/ASAP.2018. 8445101[12] Jorge Gomez、Saavan Patel、Syed Shakib Sarwar、Ziyun Li、Raffaele Capoccia、Zhao Wang、Reid Pinkham、Andrew Berkovich、Tsung-Hsun Tsai、Barbara De Salvo 等。2022。面向 AR/VR 设备的分布式传感器上计算系统：一种用于功耗估算的半解析仿真框架。*arXiv*预印本 *arXiv:2203.07474*（2022）。[13] Kazushige Goto 和 Robert A. van de Geijn。2008。高性能矩阵乘法剖析。*ACM*数学软件汇刊，第12号文章（2008），25页。[14] Kazushige Goto 和 Robert A. van de Geijn。2008。高性能矩阵乘法剖析。*ACM*数学软件34，3（2008年5月），12:1–12:25。https://doi.org/ 10 1145 1356052 1356053 。/。Mark Hill 和 Vijay Janapa Reddi。...。[15] Gables：移动片上系统的 Roofline2019模型。载于*2019*年*IEEE*高性能计算机体系结构国际研讨会（*HPCA*）。317 330 10 1109 2019 00047 –。https://doi.org/。/HPCA。...。[16] H. T. Kung、Vikas Natesh 和 Andrew Sabot2021。...。CAKE：使用恒定带宽分块的矩阵乘法。载于高性能计算、网络、存储与分析国际会议论文集（*SC '21*）。计算机协会，美国纽约州纽约，1–14。https://doi.org/ 10 1145 3458817 3476166 。/。H. T. Kung、Vikas Natesh 和 Andrew Sabot。[17] 2021。CAKE：使用恒定带宽分块的矩阵乘法。载于高性能计算、网络、存储与分析国际会议论文集（美国密苏里州圣路易斯）（*SC '21*）。计算机协会，美国纽约州纽约，第 85, 14 10 1145 3458817 3476166 页。https://doi.org/。/。Liangzhen Lai、Naveen Suda 和 Vikas Chandra。[18] 2018。CMSIS-NN：适用于 Arm Cortex-M CPU 的高效神经网络内核。*arXiv:1801.06601 [cs]*（1月 2018 1801 06601 [ ]）。arXiv：。cs Rui Li、Aravind Sukumaran-Rajam、Richard Veras、Tze Meng Low、Fabrice Rastello、Atanas Rountev 和 P. Sadayappan。[19] 2019。张量缩并的缓存解析建模与分块尺寸优化。载于高性能计算、网络、存储与分析国际会议论文集（美国科罗拉多州丹佛）（*SC '19*）。计算机协会，美国纽约州纽约，第 74, 13 10 1145 3295500 3356218 页。https://doi.org/。/。Ji Lin、Wei-Ming Chen、Yujun Lin、John Cohn、Chuang Gan 和 Song Han。[20] 2020。MCUNet：面向物联网设备的微型深度学习。载于第*34*届神经信息处理系统国际会议论文集（*NIPS'20*）。Curran Associates Inc.，美国纽约州雷德胡克，11711–11722。[21] Bradley McDanel、Sai Qian Zhang、HT Kung 和 Xin Dong。2019。利用二次幂权重加速卷积神经网络的全栈优化与 FPGA验证。载于*ACM*超算国际会议论文集。449–460。[22] Monsoon Solutions, Inc. [无日期]。低电压功率监视器文档。https://www.msoon.com/lvpm-product-documentation。[23] Google Research。2020。深度学习矩阵集合。https://github.com/google-research/google-research/tree/master/sgk。[24] Davide Rossi、Francesco Conti、Andrea Marongiu、Antonio Pullini、Igor Loi、Michael Gautschi、Giuseppe Tagliavini、Alessandro Capotondi、Philippe Flatresse 和 Luca Benini。2015。PULP：面向下一代物联网应用的并行超低功耗平台。载于*2015*年*IEEE Hot Chips*第*27*届研讨会（*HCS*）。IEEE计算机学会，1–39。[25] Tyler M. Smith 和 Robert A. van de Geijn。2019。MOMMS 矩阵乘法算法系列。*arXiv:1904.05717 [cs]*（2019年4月）。arXiv: 1904 05717 [ ]。cs Stanislava Soro。...。面向泛在边缘人工智能的 TinyML。[26] 2021。https://doi.org/10.48550/ ARXIV.2102.01255[27] Aravind Vasudevan、Andrew Anderson 和 David Gregg。2017。使用通用矩阵乘法实现并行多通道卷积。载于*2017*年*IEEE*第*28*届应用专用系统、架构与处理器国际会议（*ASAP*）。19 24 10 1109 2017 7995254 –。https://doi.org/。/ASAP。...。[28] Pete Warden。2015。...。为什么通用矩阵乘法是深度学习的核心。[29] Samuel Williams、Andrew Waterman 和 David Patterson。2009。Roofline：面向多核架构的直观可视化性能模型。*ACM*通讯 52, 4 2009 65 76 10 1145 1498765 1498785 （），–。https://doi.org/。/。[30] Zhang Xianyi、Wang Qian 和 Zaheer Chothia。2012。...。Openblas。网址：*http: //xianyi.github.io/OpenBLAS*（2012），88。[31] Vinson Young、Sanjay Kariyappa 和 Moinuddin K. Qureshi。2019。为通用内存系统实现透明内存压缩。载于*2019*年*IEEE*高性能计算机体系结构国际研讨会（*HPCA*）。570–581。https://doi.org/10.1109/HPCA.2019.00010