基于SRAM的通用存算一体架构平台在物联网中的应用

曾剑敏[①] 张 章*[①] 虞志益[②] 解光军[①]

[①](合肥工业大学电子科学与应用物理学院 合肥 230601)

[②](中山大学微电子科学与技术学院 珠海 519082)

摘 要：最近，存算一体(IMC)架构引起了广泛关注，并被认为有望成为突破冯诺依曼瓶颈的新型计算机架构，特别是在数据密集型(data-intensive)计算中能够带来显著的性能和功耗优势。其中，基于SRAM的IMC架构方案也被大量研究与应用。该文在一款基于SRAM的通用存算一体架构平台——DM-IMCA的基础上，探索IMC架构在物联网领域中的应用价值。具体来说，该文选取了物联网中包括信息安全、二值神经网络和图像处理在内的多个轻量级数据密集型应用，对算法进行分析或拆分，并将关键算法映射到DM-IMCA中的SRAM中，以达到加速应用计算的目的。实验结果显示，与基于传统冯诺依曼架构的基准系统相比，利用DM-IMCA来实现物联网中的轻量级计算密集型应用，可获得高达24倍的计算加速比。

关键词：物联网；超越冯诺依曼架构；存算一体；计算型SRAM

中图分类号：TN47 **文献标识码**：A **文章编号**：1009-5896(2021)06-1574-13

DOI: 10.11999/JEIT210010

Applications of Generic In-memory Computing Architecture Platform Based on SRAM to Internet of Things

ZENG Jianmin[①] ZHANG Zhang[①] YU Zhiyi[②] XIE Guangjun[①]

[①](*School of Electronic Science and Applied Physics*, *Hefei University of Technology*, *Hefei* 230601, *China*)

[②](*School of Microelectronics Science and Technology*, *Sun Yat-sen University*, *Zhuhai* 519082, *China*)

Abstract: In-Memory Computing (IMC) architectures have aroused much attention recently, and are regarded as promising candidates to break the von Neumann bottleneck. IMC architectures can bring significant performance and energy-efficiency improvement especially in data-intensive computation. Among those emerging IMC architectures, SRAM-based ones have also been extensively researched and applied to many scenarios. In this paper, IoT applications are explored based on a SRAM-based generic IMC architecture platform named DM-IMCA. To be specific, the algorithms of several lightweight data-intensive applications in IoT area including information security, Binary Neural Networks (BNN) and image processing are analyzed, decomposed and then mapped to SRAM macros of DM-IMCA, so as to accelerate the computation of these applications. Experimental results indicate that DM-IMCA can offer up to 24 times performance speed-up, compared to a baseline system with conventional von Neumann architecture, in terms of realizing lightweight data-intensive applications in IoT.

Key words: Internet of Things (IoT); Beyond von Neumann architecture; In-Memory Computing (IMC); Computational SRAM

1 引言

随着移动互联网、云计算、物联网和人工智能等技术的快速发展，我们正全面步入大数据时代。大数据最重要的特征之一，也是大数据时代社会所面临最大的一个挑战：有海量的数据需要处理[1]。因为在大数据时代，包括社交媒体、生物医疗、气象、交通和航空航天等在内的多种领域均会产生海量的数据。例如当前国际射电天文界最重要的大型

收稿日期：2021-01-05；改回日期：2021-04-09；网络出版：2021-04-30

*通信作者： 张章 zhangzhang@hfut.edu.cn

基金项目：国家自然科学基金(U19A2053, 61674049)，中央高校基本科研基金(JZ2020YYPY0089)，中国科学院红外成像材料与器件重点实室开放课题(IIMDKFJJ-19-04)

Foundation Items: The National Natural Science Foundation of China(U19A2053, 61674049), The Fundamental Research Funds for Central Universities (JZ2020YYPY0089), The Key Laboratory of CAS (IIMDKFJJ-19-04)

望远镜项目之一——平方千米阵(Square Kilometer Array, SKA)[2]，每年可以产生300 PB容量的数据[3]；一架波音喷气式客机飞行一小时就能产生约21 TB容量的数据[4]。社会所产生的数据是呈指数式爆炸增长的[5]，据全球知名数据公司IDC (International Data Corporation)发布的《数据时代2025》白皮书报告显示，到2025年，全球每年产生的数据将从2018年的33 ZB增长到175 ZB，相当于全球每天产生491 EB的数据[6]。

大数据的出现与繁荣发展使得数据处理的重心逐渐从以计算为中心转移到以数据为中心[7]，即数据处理任务或应用从计算密集型转移为数据密集型。而由于存储墙[8]和带宽墙[9]等原因，当前采用冯诺依曼架构设计的计算机系统在数据密集型计算中表现出的性能瓶颈和低能效等缺点日益凸显。因此，为了解决这些问题，新的计算机架构，特别是超越冯诺依曼(beyond von Neumann)架构，亟待提出。

近年来，存算一体(In-Memory Computing, IMC)架构引起了研究人员的广泛关注，并被认为是一种有望成为突破冯诺依曼瓶颈的新计算机架构范式。存算一体的核心思想是使得计算单元和存储单元尽量靠近，甚至融合为一体[10]。近期前沿文献中实现IMC架构的方式有多种，例如基于新兴的3D堆叠封装[11–13]或者忆阻器等非易失存储器件[14–16]来实现。然而，由于3D堆叠以及非易失存储等新兴技术并不十分成熟，基于这些技术设计的IMC架构短时间很难得到广泛应用。因此，许多文献[17–24]逐渐基于技术成熟的SRAM来探索和设计IMC架构，并证明基于SRAM的IMC架构在实现数据密集型应用时能够带来显著的性能和能效提升。例如，文献[20]为本文作者所设计的一种基于SRAM实现的通用IMC架构平台——DM-IMCA。为弥补现有文献中几乎所有基于SRAM来实现的IMC架构均面向如神经网络等专用目的而设计的缺憾，DM-IMCA能够在其内部SRAM中进行大部分的逻辑运算和算术运算，因此具有广泛的通用性，并且具有较高的潜力和价值。

为了充分挖掘DM-IMCA的应用潜力和价值，本文探索了该平台在物联网领域中的应用。详细来说，本文选取了包括物联网中信息安全、深度神经网络以及图像处理在内的若干轻量型数据密集型应用，对相关算法进行分析或者拆分，并把算法的关键部分映射到DM-IMCA的SRAM中进行计算，以达到加速应用计算或者降低功耗的目的。

2　基于SRAM的存内逻辑计算原理

在传统SRAM的1个读操作周期内，共用同1对位线的若干个存储单元中，只有其中1个会被选中读取。如果对位于SRAM同一列的两个或者多个存储单元同时进行读取，最后位线上电平所代表的逻辑值将是被打开的单元所存储比特的相与结果[17]。图1展示了在SRAM列内实现计算的基本原理。晶体管M1—M6和M7—M12分别构成了两个6管SRAM存储单元$\mathrm{MC_A}$和$\mathrm{MC_B}$，且这两个存储单元位于同1对位线上，$Q_\mathrm{A}/\mathrm{QB_A}$和$Q_\mathrm{B}/\mathrm{QB_B}$分别为$\mathrm{MC_A}$和$\mathrm{MC_B}$上的存储节点，且记$Q_\mathrm{A}$=$A$和$Q_\mathrm{B}$=$B$。记位线BL和BLB上的逻辑值分别为$F$和$F_\mathrm{B}$。当$A$=$B$=1时，读取$\mathrm{MC_A}$和$\mathrm{MC_B}$中任何一个单元将会使已经预充电的BLB放电，而BL的电位将会保持不变。若将$\mathrm{MC_A}$和$\mathrm{MC_B}$的字线同时打开，位线BLB会同时通过通路①和通路②来放电，而位线BL将会保持高电平。观察容易发现，位线BL和BLB上的逻辑值分别是打开的两个存储单元$\mathrm{MC_A}$和$\mathrm{MC_B}$中值的“与”和“或非”结果，即$F=A\cdot B$和$F_\mathrm{B}=\overline{A}\cdot\overline{B}=\overline{A+B}$，如图1(a)所示。若在位线上增加一个灵敏放大器(Sense Amplifier, SA)，并将两个SA的其中一端接至合适的公共参考电压，那么便可以在这两个SA的输出端分别得到F和F_B这两个逻辑结果，如图1(b)所示。事实上，位线上的逻辑运算不仅对于两个存储单元成立，对于同一位线上的多个存储单元也适用。记同一对位线BL和BLB上所挂载单元中所存储的内容分别为$A_1, A_2, \cdots, A_n$，两个SA输出的逻辑分别为F和F_B，那么理论上可以得到逻辑运算关系式

$$F=A_1\cdot A_2\cdots A_3 \tag{1}$$

和

$$F_\mathrm{B}=\overline{A_1}\cdot\overline{A_2}\cdots\overline{A_n}=\overline{(A_1+A_2+\cdots+A_n)} \tag{2}$$

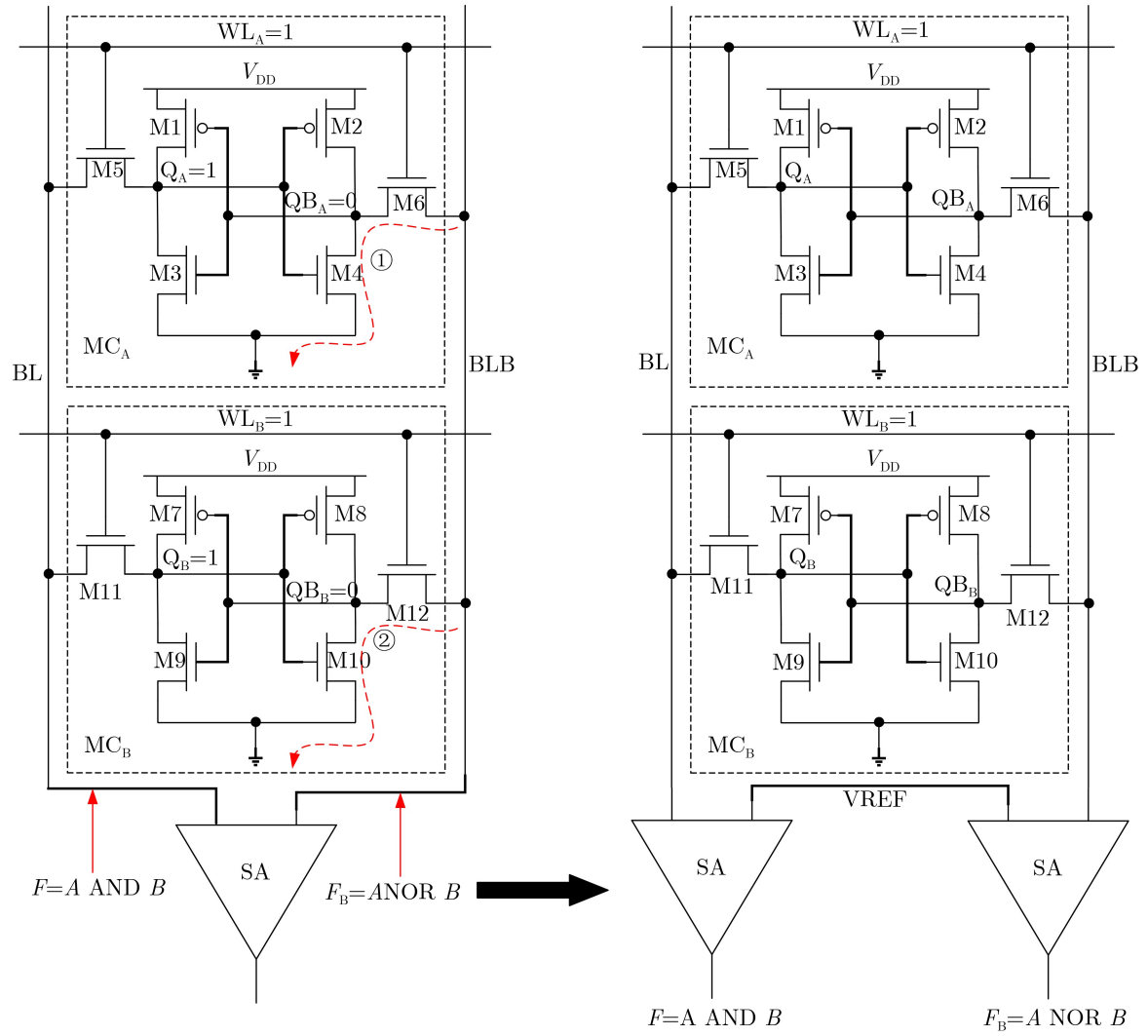
即可以利用SRAM位线的“线与”特性来实现多输入与门和或非门。

3　DM-IMCA：基于SRAM的通用IMC架构平台

3.1　DM-IMCA简介

DM-IMCA[20]是一个基于SRAM的通用IMC架构平台，其硬件架构如图2所示。DM-IMCA主要由1个6级流水精简指令集处理器核、1个指令存储器、1个存内计算协处理器——IMC-CP，以及由若干SRAM模块组成的数据存储器组成。其中处理器核是基于一款开源、具有经典5级流水且兼容MIPS32架构的低功耗轻量处理器核OpenMIPS进行裁剪与改进而来的。

数据存储器由若干常规SRAM模块与计算型SRAM——IMC-SRAM组成。IMC-SRAM是一款



(a) 位线逻辑运算 (b) 双SA输出

图 1 SRAM列内逻辑运算示意图

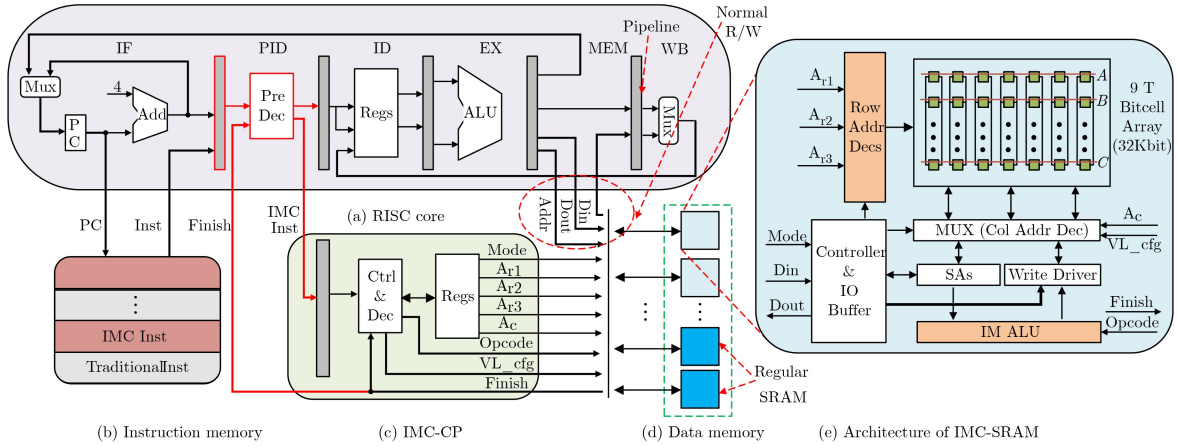


图 2 DM-IMCA硬件架构图

融合存储和计算为一体的SRAM，其硬件架构如图2(e)所示。IMC-SRAM是在如图1所示的电路原理基础之上，将传统6管单元换成9管单元，以消除6管单元所带来的读写互扰以及进行存内计算时的问题。此外，IMC-SRAM中还额外加入了少许逻辑门，用于实现除“与”和“或非”逻辑之外的其他运算。与已有文献相比，IMC-SRAM可以支持更多类型的运算，例如加法、移位运算等。此外，

IMC-SRAM支持存内向量计算。如图2(e)所示，将向量操作数$\boldsymbol{A}$和$\boldsymbol{B}$沿字线方向对齐存储，那么只需1次操作，便可对存储阵列中某一行的$\boldsymbol{A}$和$\boldsymbol{B}$分量计算完，然后将计算结果回写至同样与$\boldsymbol{A}$和$\boldsymbol{B}$对齐存储的向量$\boldsymbol{C}$中。在IMC-SRAM进行存内计算时，数据存储器中的常规SRAM仍然可以进行数据的读写操作。

IMC-CP是一个为存内计算设计的轻量协处理器，主要作用是对处理器核传送过来的IMC指令进行译码，然后根据译码后的信息对IMC-SRAM进行模式配置及控制存内计算操作，包括计算并配置操作数地址以及管理IMC-SRAM中的存内计算过程。IMC-CP中还包括几个重要的可配置的状态寄存器：

(1)R0：用来保存下一次存内计算的矢量长度信息；

(2)R1-R3：用来保存IMC-CP对IMC指令译码后得到的源操作数和目的操作数的列地址；

(3)Rm：用来保存系统当前工作模式；

(4)Rn：用来保存系统中用以存内计算的IMC-SRAM模块的数量信息；

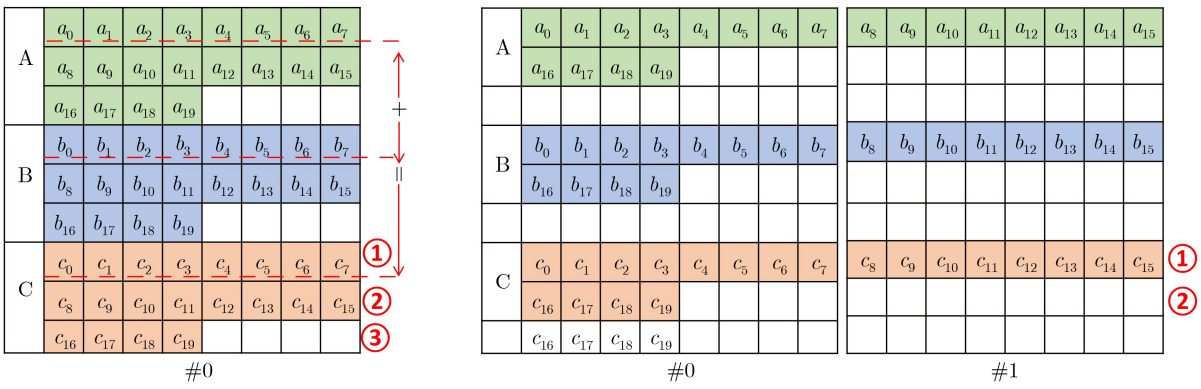
(5)Rv：保存当前IMC计算指令中的剩余矢量长度信息；

(6)Roc：保存存内计算的操作码信息。

DM-IMCA还拥有一个重要的机制——存内向量计算自动化及自适应。图3所示的实例展示了存内向量计算示意图及其自动化及自适应机制的基本原理。假设有3个长度为20的数组，其中$\boldsymbol{A}=\{a_0, a_1, \cdots, a_{19}\}$，$\boldsymbol{B}=\{a_0, b_1, \cdots, b_{19}\}$，$\boldsymbol{C}=\{c_0, c_1, \cdots, c_{19}\}$，且$\boldsymbol{C}=\boldsymbol{A}+\boldsymbol{B}$。如果只配置一个IMC-SRAM模块参与存内计算，则需要进行3次存内计算才能完成$\boldsymbol{C}=\boldsymbol{A}+\boldsymbol{B}$，图3(a)中带圈的数字表示数据存储器采用一个IMC-SRAM模块的情况下每个存内计算阶段所完成的向量分量计算。只需要一条存内计算指令，IMC-CP便可以自动控制IMC-SRAM完成多次存内计算操作，节省了指令存储器空间以及处理器流水线，也加快了存内向量计算的速度。如果配置更多的IMC-SRAM模块参与存内计算，单次存内计算操作所完成的向量长度将会成倍数增加。若数据存储器采用两个IMC-SRAM模块，对于同样的$\boldsymbol{C}=\boldsymbol{A}+\boldsymbol{B}$计算，只需要两个阶段便可以完成，如图3(b)所示。只需将参与存内计算的IMC-SRAM模块数量配置至寄存器Rn中，IMC-SRAM便可以通过Rn的值对多个模块的存内计算进行控制，这就是DM-IMCA的存内向量计算自适应机制。

在DM-IMCA中，运行着两套指令：传统MIPS32指令和IMC指令(IMC instruction)。前者通过完整的处理器核流水线来执行，后者在处理器核的指令预译码流水级之后，传送至IMC-CP进行二级译码，再由IMC-SRAM进行执行。两套指令长度一致，因此，并不需要设计两组不同的指令存储器，可以将传统MIPS32指令和IMC指令混合存储于同一个指令存储器中。这样做的好处是便于对系统进行程序设计，达到真正的传统指令和IMC指令混合编程的目的。对于两种指令的识别，将会交由处理器核的指令预译码流水级来进行处理。指令存储器采用常规的SRAM(通过工艺厂商提供的存储编译工具Memory Compiler生成)来进行设计。

DM-IMCA支持双工作模式：普通模式和IMC模式。当DM-IMCA工作于普通模式时，数据的运算由处理器核的流水线来完成，此时IMC-SRAM与传统SRAM一样，其功能是进行数据的存取，此时DM-IMCA表现得与传统冯诺依曼架构一样；当DM-IMCA工作于IMC模式时，IMC-SRAM可以进行存内计算，数据无需传送至流水线进行处理，此时DM-IMCA表现出与传统冯诺依曼架构不一样的特性。因此，DM-IMCA是一种超越传统冯诺依曼的混合架构。



(a) 调用1个IMC-SRAM模块　　(b) 调用2个IMC-SRAM模块

图 3 存内向量计算自动化及自适应示意图

3.2 IMC指令集

DM-IMCA包含一个专门为存内计算设计的指令集——IMC指令集。DM-IMCA中的处理器核经过改进，将其中不需要的部分指令去除，将腾出的指令空间用于IMC指令的编码。IMC指令的高5位为识别码，其中前3位(110)为1级识别码，用于区分IMC指令和传统MIPS32指令。1级识别码在DM-IMCA处理器核流水线中的预译码级被处理，并不会被传送至协处理器IMC-CP中。因此，处理器核传送给IMC-CP的指令实际上只有29位(IMC指令的低29位)。识别码中的后两位为2级识别码，用于分区某条IMC指令的类型。各类指令的指令格式也分别在表1中列出。

表 1 IMC指令集编码格式

指令	字段	名称
储配置指令： memCfg Rn	31–27	op (11001)
	26–4	Reserved
	3–0	rn
地址配置指令： addrCfg R3, R2, R1	31–27	op (11000)
	26–20	r3
	19–13	r2
	12–6	r1
	5–0	Reserved
计算指令： opcode vl (vl: vector length，矢量长度)	31–27	op (11010)
	26–23	function
	22–15	vl
	14–0	Reserved

按照功能划分，IMC指令可以分为两大类：配置指令和计算指令，如表2所示。其中配置指令又可以分为存储配置指令和地址配置指令。存储配置指令的作用是对数据储存器中用于存内计算的IMC-SRAM宏模块数量进行配置。具体配置过程是将指令中rn字段所包含的信息写入IMC-CP中的寄存器Rn中。地址配置指令用于配置存内计算的源操作数和目的操作数的行地址信息。地址配置指令中的r1和r2字段所包含的内容为存内计算源操作数的行地址信息，r3字段所包含的内容为存内计算目的操作数的行地址信息。地址配置指令的配置过程为分别将r1字段和r2字段内容写入IMC-CP的寄存器R1和R2中，将r3字段内容写入IMC-CP的寄存器R3中。计算指令的作用是控制IMC-SRAM的存内计算，包括存内计算的功能类型和向量长度。计算指令中的function字段包含了存内计算的操作码(功能码)，用于IMC-SRAM选择特定的存内计算功能，vl字段包含了相应存内计算操作的向量长度信息。

表 2 IMC指令集

指令类型		指令	操作/功能
配置指令	存储配置	memCfg	配置寄存器Rn
	地址配置	addrCfg	配置存内计算地址寄存器R1～R3
计算指令	逻辑计算	mand	$c=a\&b$
		mor	$c=a\|b$
		mxor	$c=a\oplus b$
		mnor	$c=\sim(a\|b)$
		mnand	$c=\sim(a+b)$
		mnot	$c=\sim a$
	算术计算	madd	$c=a+b$(有符号)
		maddu	$c=a+b$(无符号)
		mop	$c=-a$
		minc	$c=a+1$
		mdec	$c=a-1$
		msl	$c=a\ll 1$
		msr	$c=a\gg 1$
	存储操作	mcopy	$c=a$

3.3 DM-IMCA的工作流程

首先，在进行存内计算操作之前(一般为系统启动或者复位后)，需要执行一条存储配置指令memCfg来对IMC-CP中的寄存器Rm进行配置，以确定数据存储器中参与存内计算的IMC-SRAM宏模块数量。由于Rm在系统启动或者复位后被置为1，因此如果不对寄存器Rm进行配置，而又需要多个IMC-SRAM模块参与存内计算，存内计算将无法正确进行。

完成一个存内计算操作需要DM-IMCA连续执行两条IMC指令。第1条为地址配置指令addrCfg，第2条则是一条计算指令。第1条指令执行所包含的内容为第2条指令将执行的存内计算操作的源操作数和目的操作数的行地址信息。第1条指令被IMC-CP执行完后，IMC-CP中的寄存器R1-R3将会被写入相应的操作数行地址信息，并且寄存器Rm被写入“1”，表示IMC-CP被切换到IMC模式。在随后的一个周期内，IMC-SRAM的行地址信息和工作模式将会被IMC-CP进行配置。第2条计算指令的作用则是让IMC-SRAM开始启动存内计算操作。计算指令被IMC-CP执行后，计算指令中所包含的存内计算操作码和向量长度信息会被IMC-CP分别写入寄存器Roc和Rv中，随后IMC-SRAM中的存内计算流程正式开始。当IMC-SRAM完成存

内计算后，将寄存器Rm清零，将IMC-CP切换至普通工作模式，并且通知IMC-SRAM和处理器核切换至同样的工作模式。

下面以图3中的存内向量加法为例来说明存内计算的指令执行过程。为简化起见，将IMC-SRAM的起始地址设为0。对于图3(a)中的实例，只有一个IMC-SRAM模块参与存内计算，那么需要通过存储配置指令将Rn设置为“1”(如果系统刚启动或者复位，也可以不设置)。向量(数组)$\boldsymbol{A}$, $\boldsymbol{B}$和$\boldsymbol{C}$的行地址分别为0, 3和6，存内加法计算的向量长度vl为20，那么完成$\boldsymbol{C}=\boldsymbol{A}+\boldsymbol{B}$这个向量加法需要执行以下几条指令：

(1) memCfg 1
(2) addrCfg 6, 3, 0
(3) maddu 20

与图3(a)相比，图3(b)中实例的区别在于调用了2个IMC-SRAM模块参与存内计算。因此，完成同样的存内计算，除了Rn的配置不同，其余均与图3(a)中的实例一致。

4　DM-IMCA在物联网信息安全中的应用

随着物联网应用的越来越广泛，其安全性也变得更加重要。因此，非常有必要对所传输的数据进行加密来确保物联网的信息安全。利用加密算法可以有效提高物联网中设备通信时的数据安全。本节将以两个轻量级加密算法为例，介绍DM-IMCA在信息安全中的应用。这两个加密算法分别是一次性密码(One-Time Pad, OTP)[25]算法和哈希算法[26](或称为散列算法)。

4.1　基于DM-IMCA的OTP加密

在OTP加密技术中，信息发送方使用与信息长度一致的密钥对信息进行加密，信息接收方在接收到加密信息后使用与加密时相同的密钥进行解密。由于加解密使用了相同的密钥，因此OTP属于对称加密类型。为保障加密的有效性和安全性，加解密使用的密钥必须为一次性的，即一个密钥只能使用一次。此外，加解密所使用的密钥生成方式应该做到最大限度的随机性。由于OTP技术的密钥长度需要和被加密信息一致，因此其对大文件的加密效率较低，而比较适用于短信息加密。OTP加密的核心过程非常简单，只需对信息内容和一次性密钥做按位异或操作，运算结果便是加密后的内容，如表3所示。

DM-IMCA原生支持异或操作，因此OTP加密过程可直接通过使用IMC指令来实现。将表3中的数组$\boldsymbol{K}$, $\boldsymbol{P}$和$\boldsymbol{C}$在IMC-SRAM中映射的行地址分别设置为32, 40和48，则其字节地址分别为0×400, 0×500和0×600。通过DM-IMCA存内计算的方法来实现表3中算法的核心代码如下：

(1) addrCfg 48, 40, 32
(2) mxor N

此处N为各数组的字长信息。如果通过基准系统来实现同样的OTP算法，其汇编代码如下：

(1) li \$t0, 0
(2) loop:
(3) lw \$t4, 0x400(\$t0)
(4) lw \$t5, 0x500(\$t0)
(5) xor \$t6, \$t4, \$t5
(6) sw \$t6, 0x600(\$t0)
(7) addiu \$t0, \$t0, 4
(8) blt \$t0, N_{b}, loop

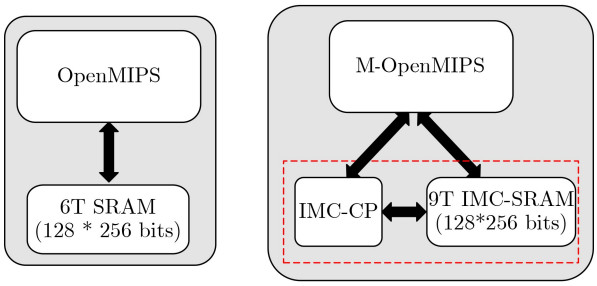
其中，代码中的$N_{\mathrm{b}}=N/4$，因为对于MIPS汇编来讲，地址是以字节而不是字为单位计算的。通过上述OTP算法在不同系统中实现的汇编代码对比，也可以看出，相比于传统的冯诺依曼架构的基准系统，DM-IMCA的代码要简洁许多，可以节省大量指令存储器空间。

表 3　OTP算法

输入: plaintext $P[N]$, key $K[N]$, N为数组$\boldsymbol{P}$和$\boldsymbol{K}$的字长
输出: ciphertext $C[N]$
(1)　for (unsigned $i=0$; $i<N$; i++) do
(2)　$C[i]=P[i]\oplus K[i]$;
(3)　end for

为了测试与对比，本文搭建了一个测试平台，如图4所示。其中图4(a)是一个作为对照组的基准系统(baseline system)，主要由OpenMIPS、1块大小为32 kbit (128行×256列)的传统6管SRAM宏单元；图4(b)是用于测试的DM-IMCA系统，主要由一个本文改进后的6级流水处理器核M-OpenMIPS、1个大小为32 kbit (128行×256列)的IMC-SRAM模块以及协处理器IMC-CP组成。基准系统及DM-IMCA测试系统均采用由存储编译器(memory compiler)生成的6管SRAM模型作为指令存储器。该平台将用于本文所有应用的测试。

分别选择长度为256 bit($N=8$字)和1024 bit ($N=32$字)的密钥，分别在DM-IMCA和基准系统中进行加密实验。在加密前，明文数组$\boldsymbol{P}$、密钥数组$\boldsymbol{K}$和密文数组$\boldsymbol{C}$在IMC-SRAM中的存储方式类似图3的向量$\boldsymbol{A}$, $\boldsymbol{B}$和$\boldsymbol{C}$。由于在实验中，DM-IMCA只采用了一个IMC-SRAM宏模块，因此$N=8$时，各数组分别需要占用IMC-SRAM一行的存储空



(a) 基准系统(Baseline) (b) DM-IMCA实验系统

图 4 测试平台

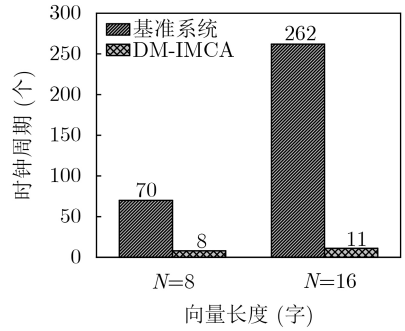


图 5 测试OTP加密所花费的时间对比

间，而$N=32$时，各数组占用的IMC-SRAM的存储空间则为4行。分别统计通过DM-IMCA和基准系统进行OTP加密实验所花费的时钟周期数，如图5所示。结果显示，与基准系统相比，利用DM-IMCA的存内运算来处理256 bit和1024 bit长度的OTP加密，分别可以获得8.75和23.8倍的运算加速比。

4.2 基于DM-IMCA的哈希函数实现

哈希算法的功能是将任意长度的信息映射到固定长度的序列中，这个序列称为哈希值(Hash code)。通常哈希值的长度比信息小得多，且哈希算法的运算是不可逆的，因此常用于信息提炼，以保障信息的真实性。因此信息发送者可以将原始信息和相应的哈希值一起发送，信息接收者通过对接收到的信息进行同样的哈希运算并将结果与接收到哈希值进行对比，来确认收到的信息是否为原始数据。这其实就是常用到的文件校验功能。另一个哈希应用的实例是网站用户登录机制中的密码找回。当用户忘记自己注册的某个网站的登录密码，申请找回密码时，为了保护用户的信息安全，网站管理方不是将登录密码明文发给用户，而是将登录密码的哈希值发送给用户，这样就可以避免密码遭到泄露。除此之外，哈希算法的应用还非常广，例如数字签名、协议鉴权等。随着物联网和区块链的发展，哈希算法在该领域也有着许多应用[27]。

哈希算法实际上是一类算法，而不单指某个算法，常见的MD5和SHA, 以及HMAC等均属于哈希算法的范畴。一般把产生哈希值的函数称为哈希函数，按照哈希值的产生方式不同，哈希函数可以分为加法哈希、乘法哈希、位运算哈希等几种经典的函数。这些基础的哈希函数组合还可以构成更复杂的哈希函数。本节将选取加法哈希在DM-IMCA以及基准系统中进行实现。加法哈希的描述如表4所示，其中key和len分别表示输入的字符串和字符串长度，prime是任意的质数。

从表4可以看出，加法哈希函数运算的核心部分实质是一个加法树，其规模随着字符串长度的增加而越来越庞大。这样的运算在基准系统中实现只能是以串行的方式逐个进行累加，显然效率较低，耗时较长。在DM-IMCA中，可以将字符串数组映射于IMC-SRAM中，按存内计算的存储要求排列，分批次进行运算。假设字符串长度len = 128时，字符串数组key在IMC-SRAM中的起始地址为0×400(行地址为32)。先将数组key的元素对半分为两个新的数组$\boldsymbol{K}1$和$\boldsymbol{K}2$，并分别按如图6所示的方式依次存储于IMC-SRAM中(需要注意的是，由于当前IMC-SRAM不支持字符型数据的操作，因此需将char型数据转换为unsinged int型)。$\boldsymbol{K}1$和$\boldsymbol{K}2$通过存内加法进行各元素相加，并将结果存于$\boldsymbol{K}1$中，即

$$K1[i] \leftarrow K1[i] + K2[i], i \in [0,63] \tag{3}$$

按式(3)完成第1阶段运算后，接着用同样的方法将数组$\boldsymbol{K}1$对半分为两部分，分别记为$\boldsymbol{K}3$和$\boldsymbol{K}4$，再通过存内计算执行如下操作：$K3[i] \leftarrow K3[i]+K4[i], i \in [0,31]$。以此类推，反复进行同样的数组对半拆分操作及存内加法运算，直到无法使用存内运算进行加法操作，将剩下的元素取出至处理器核中进行相加。对于字符串长度len = 128的实例，存内矢量加法运算的相关汇编代码如下：

(1) addrCfg 32, 40, 32
(2) maddu 64 // $K1[i] \leftarrow K1[i] + K2[i], i \in [0,63]$
(3) addrCfg 32, 36, 32
(4) maddu 32 // $K3[i] \leftarrow K3[i] + K4[i], i \in [0,31]$

表 4 加法哈希算法

输入: char *key, uint32_t len, uint32_t prime
输出: result
(1) uint32_t hash, i
(2) for (hash = len, $i = 0$; $i <$ len; i++) do
(3) hash +=key[i]
(4) end for
(5) return (hash % prime)

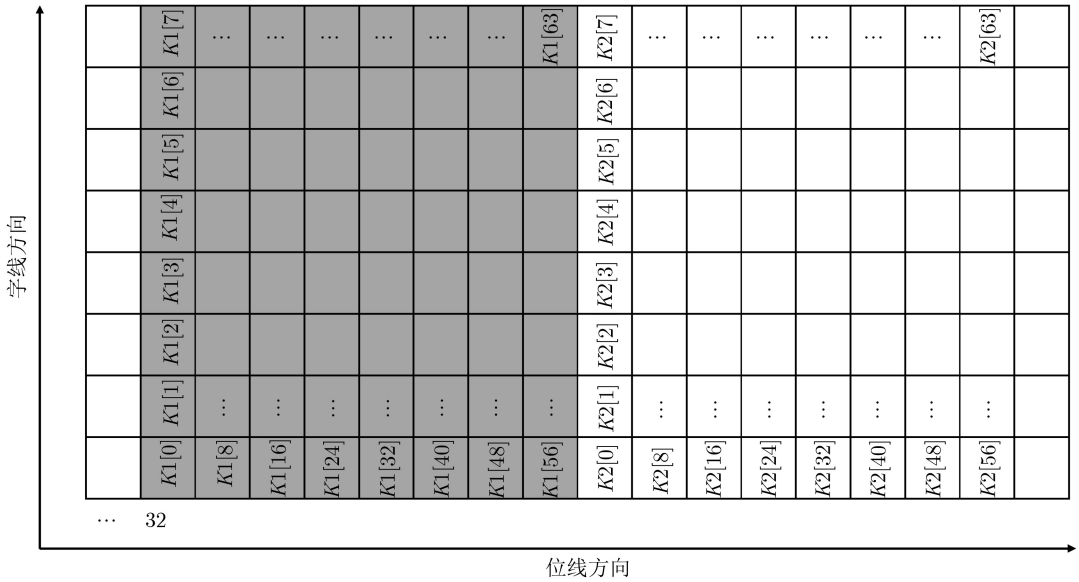


图 6 加法哈希函数运算中字符串在IMC-SRAM中的映射

(5) addrCfg 32, 34, 32

(6) maddu 16

(7) addrCfg 32,33, 32

(8) maddu 8

分别取长度len为256和512的字符串使用加法哈希算法在基准系统和DM-IMCA中进行试验，并统计各自所消耗的时钟周期数，其结果如图7所示。结果显示，与基准系统相比，使用DM-IMCA的存内运算功能进行长度为256和512位字符的加法哈希加密运算分别可以获得约6.5倍和12.2倍的运算加速比。通过图7还可以看出，随着数日字符串长度翻倍，基准系统所需的加法哈希运算时间也几乎翻倍，而使用DM-IMCA来运算增加的时间非常少。因此，字符串越长，使用DM-IMCA进行加法哈希运算越有优势。

5　DM-IMCA在深度神经网络中的应用

近些年来，随着集成电路工艺的发展，计算机的算力获得了巨大提升。此外，在信息化时代，社会所产生的数据呈爆炸式增长。这些因素极大地促进了人工智能技术的快速发展，从而使人工智能得到了极广泛的应用，包括热门的计算机视觉、语音识别，以及自然语言处理[28]。在众多的人工智能模型中，卷积神经网络(Convolutional Neural Networks, CNNs)的发展尤其成功，例如著名的LeNet-5[29], VGGNet[30]和ResNet[31]，均属于CNN。然而，CNN对存储和计算资源有着巨大的需求，例如VGG-16网络需要存储138 MB的参数，为了对一幅大小为224×224的图像进行分类，需要进行高达15.5 G次的浮点精度MAC(Multiply-Accumulate)运算；ResNet-50也仍然需要存储25.5 MB的参数，对同样大小的图片进行分类，需要高达3.9 G次的浮点精度MAC运算。对于面积和功耗等资源紧张型的场景，如嵌入式设备或移动终端，以及物联网中的许多边缘设备，这样的原生网络实在过于庞大，不便于部署。庆幸的是，Courbariaux等人通过研究证明可以同时将权值和神经元激活(neuron activations)二值化，即同时减小到1 bit精度，同时不会遭受明显的分类精度损失[32,33]。例如，可以采用如式(4)所示的算法进行二值化。

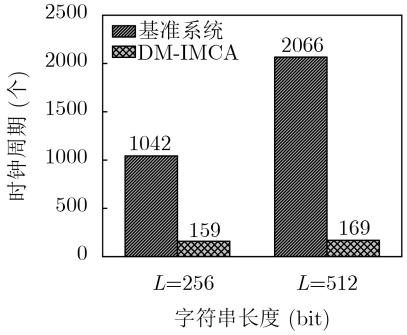


图 7 测试加法哈希算法进行加密所耗费的时间对比

$$x^{\mathrm{b}} = \mathrm{Sign}(x) = \begin{cases} +1, x \geq 0 \\ -1, x < 0 \end{cases} \tag{4}$$

其中，x^{b}是二值化后的变量(权值或者激活)。记卷积操作的权值和激活矩阵展开成向量后分别为$\boldsymbol{W} = \{w_0, w_1, \cdots, w_N\}$和$\boldsymbol{a} = \{a_0, a_1, \cdots, a_N\}$，其中$N$为向量的长度。设$i = 0, 1, \cdots, N$(下文中$i$取值相同)，记$\boldsymbol{f} = \{f_0, f_1, \cdots, f_N\}$且$f_i = a_i \times w_i$。在网络二值化后，所有的$a_i$和$w_i$在–1和1二者中取值，那么二值乘法的真值表如表5所示。通过该表可以看

出，二值(–1和1)乘法关系类似于同或逻辑(XNOR)。若将a_i和w_i中所有的–1用0替换，并将变换后的权重和激活向量分别记为$\boldsymbol{W}'=\{w'_0, w'_1,\cdots,w'_N\}$和$\boldsymbol{a}'=\{a'_0, a'_1,\cdots,a'_N\}$。记$\boldsymbol{f}'=\{f'_0, f'_1,\cdots,f'_N\}$且$f_i{}'=a'_i\odot w'_i$，那么$\boldsymbol{f}'$刚好是将$\boldsymbol{f}$中的–1替换成0后的结果。因此原来的二值乘法将变换成同或运算。$\boldsymbol{a}'$和$\boldsymbol{W}'$的卷积计算等价于先进行XNOR运算求$\boldsymbol{f}'=\{f'_0,f'_1,\cdots,f'_N\}$，再计算$\boldsymbol{f}$中1的个数，其中后者称为popcount[32]，记为popcount(xnor(a'_i,w'_i))。很容易发现，$\boldsymbol{f}$中1的个数与$\boldsymbol{f}'$中的相同，$\boldsymbol{f}$中–1的个数与$\boldsymbol{f}'$中0的个数相同，那么$\boldsymbol{f}'$中–1的个数为$N-$popcount(xnor(a'_i,w'_i))，那么$\boldsymbol{a}$和$\boldsymbol{W}$的卷积运算结果可表示为

$$\boldsymbol{a}*\boldsymbol{W}=\sum_{i=0}^{N}a_i\times w_i=2\times\text{popcount}(\text{xnor}(a'_i,w'_i))-N \tag{5}$$

表 5 二值乘法真值表

a_i	w_i	f_i=a_i×w_i
–1	–1	1
–1	1	–1
1	–1	–1
1	1	1

将神经网络的权值和激活二值化可以极大地压缩参数所需的存储空间，节省硬件计算资源。此外，功耗也相应得到大幅度降低[32]。

从式(5)可以看出在二值神经网络中，计算最密集的卷积运算最后可以化为按位XNOR逻辑运算和“数1”操作。在实际的CNN网络中，MAC运算通常并行度非常大，因此在网络二值化后，也存在着并行度较大的逻辑运算，这恰好是DM-IMCA善于处理的问题。然而IMC-SRAM的存内计算并不支持直接的XNOR运算，因此有必要对式(5)做一些变换，使其可以直接映射到IMC-SRAM的存内运算中，如图8。$\boldsymbol{f}''=\{f''_0,f''_1,\cdots,f''_N\}$且$\boldsymbol{f}''_i=\sim f'_i=a'_i\oplus w'_i$，那么$\boldsymbol{f}''$中1的个数为popcount(xnor$(a'_i,w'_i)$)，且与$\boldsymbol{f}$中–1的个数相同；$\boldsymbol{f}''$中0的个数与$\boldsymbol{f}$中1的个数相同。$\boldsymbol{a}$和$\boldsymbol{W}$的卷积运算公式由式(5)变换为

$$\boldsymbol{a}*\boldsymbol{W}=\sum_{i=0}^{N}a_i\times w_i=N-2\times\text{popcount}(\text{xnor}(a'_i,w'_i)) \tag{6}$$

式(6)中的xor运算可以通过DM-IMCA中高度并行的mxor指令来实现存内计算，操作类似前述的OTP加密运算；此外，popcount操作也可以通过移位和加法等多类存内运算组合来完成。为了充分利用IMC-SRAM存储空间，首先将向量$\boldsymbol{a}$和$\boldsymbol{W}$的元素以字为单位合并后存储于整形数组中。图8所示的是当向量长度$L=256$时，激活$\boldsymbol{a}$和权值$\boldsymbol{W}$的存储映射实例。$\boldsymbol{a}$和$\boldsymbol{W}$分别映射到数组$\boldsymbol{A}$和$\boldsymbol{B}$中，并在IMC-SRAM中按行对其排列。其次，利用存内运算来完成XOR操作。随后分别通过移位、数据位屏蔽以及加法来完成popcount操作。表6描述了popcount(xnor(a'_i,w'_i))操作在IMC-SRAM中的存内运算映射，其中第(3)行描述的是XOR操作，第(4)～(11)行描述的popcount操作。第(4), (5), (6)和(8)行描述的运算分别可以通过DM-IMCA中的mxor, mand, maddu和msr指令来实现并行运算。表6中算法中涉及的数组$\boldsymbol{M}$, $\boldsymbol{C}$和$\boldsymbol{D}$在IMC-SRAM中的存储映射如图8所示。若输入数组$\boldsymbol{A}$和$\boldsymbol{B}$的长度较大，第(12)～(14)行操作还可以使用存内加法进一步进行加速，类似前面所介绍的对哈希算法中的加法树运算。

表 6 存内矢量XOR运算和popcount操作算法

输入: int $A[N]$, $B[N]$, N为数组$\boldsymbol{A}$和$\boldsymbol{B}$长度
输出: S

(1) $M[i]\leftarrow$ 0x0001, $i\in[0,N-1]$
(2) $D[i]\leftarrow 0, i\in\in[0,N-1]$
(3) $S\leftarrow 0$
(4) $A[i]\leftarrow A[i]\oplus B[i], i\in[0,N-1]$
(5) $C[i]\leftarrow A[i]\&M[i], i\in[0,N-1]$
(6) $D[i]\leftarrow D[i]+C[i], i\in[0,N-1]$
(7) for $(i=0; i<31; i{+}{+})$ do
(8) $C[i]\leftarrow A[i]\gg 1, i\in[0,N-1]$
(9) $C[i]\leftarrow A[i]\&M[i], i\in[0,N-1]$
(10) $D[i]\leftarrow D[i]+C[i], i\in[0,N-1]$
(11) end for
(12) for $(i=0; i<N; i{+}{+})$ do
(13) $S\leftarrow S+D[i], i\in[0,N-1]$
(14) end for
(15) return S

设数组$\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{M}$, $\boldsymbol{C}$和$\boldsymbol{D}$在IMC-SRAM中的行地址分别为32, 40, 48, 56和64，当$L=256$时，在DM-IMCA中实现表6中算法中的第(4)～(11)行运算的汇编代码如下：

(1) addrCfg 32, 32, 32
(2) mxor 8 // $A[i]\leftarrow A[i]\oplus B[i], i\in[0,N-1]$
(3) addrCfg 56, 32, 48
(4) mand 8 // $C[i]\leftarrow A[i]\&M[i], i\in[0,N-1]$

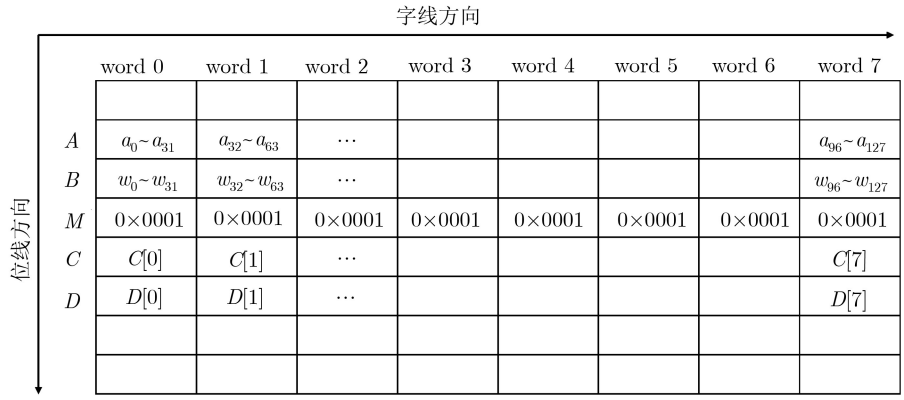


图 8 二值神经网络中激活和权重在IMC-SRAM中的映射

(5) addrCfg 64, 64, 56
(6) maddu 8 // $D[i] \leftarrow D[i] + C[i], i \in [0, N-1]$
(7) li \$t0, 0
(8) loop:
(9) addrCfg 32, 32, 32
(10) msr 8 // $C[i] \leftarrow A[i] \gg 1, i \in [0, N-1]$
(11) addrCfg 56, 32, 48
(12) mand 8 // $C[i] \leftarrow A[i] \& M[i], i \in [0, N-1]$
(13) addrCfg 64, 64, 56
(14) maddu 8 // $D[i] \leftarrow D[i] + C[i], i \in [0, N-1]$
(15) addiu \$t0, \$t0, 1
(16) blt \$t0, 31, loop

而在基准系统中实现相同的功能，各步的运算都只能使用串行操作，因此运算效率将会比在DM-IMCA中实现更低。

在实际的二值神经网络中，通常存在多维度的卷积操作，从而使参与MAC运算的两个向量长度均比较长。因此，对应的二值卷积计算中$\boldsymbol{a}$和$\boldsymbol{W}$的向量长度也较长。本节将通过实验模拟较长的二值卷积计算，从而观察DM-IMCA存内运算对其的加速效果。分别取向量$\boldsymbol{a}$和$\boldsymbol{W}$的长度$L=512$, $L=1024$和$L=2048$，分别在基准系统和DM-IMCA系统中实现，最后统计各自所消耗的时钟周期数，结果如图9所示。经计算可知，相比于基准系统，采用DM-IMCA实现长度$L=512$, $L=1024$和$L=2048$二值卷积运算，分别可以获得7.7倍、12.4倍和17.8倍的运算加速比。

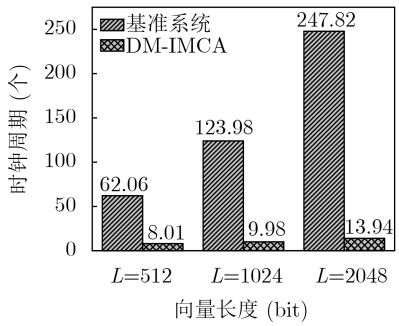


图 9 测试二进制卷积运算所花费的时钟周期数对比

6　DM-IMCA在图像处理中的应用

除了在信息安全和卷积神经网络领域，DM-IMCA在其他领域也存在着潜在的应用价值，例如传统的图像处理和视频处理领域。本节将简单介绍DM-IMCA在图像灰度化处理中的应用，以期能够对DM-IMCA在其他方面的应用起到启发作用。

在计算机图形学中，每幅图像使用一个2维数组表示，数组中的每个元素均表示一个像素点(Pixel)。根据像素点表示的不同，图像可以分为多种类型：真彩色图像、索引图像、二值图像和灰度图[34]。其中二值图像的每个像素点仅在“0”和“1”中取值，因此图像只有白和黑两种颜色。灰度图像是每个像素只有一个采样颜色的图像，每个像素点在[0, 255]范围内取值，0～255中间的每个值都代表了一个级别的灰度。因此灰度图像体现的是一幅图像中各区域的颜色深浅。真彩色图像也称为RGB图像，每个像素点由红(R)、绿(G)和蓝(B)3个分量来按不同灰度叠加构成，每个分量取值均在[0, 255]范围内。真彩色图像能够更真实地表达现实世界，因此在日常生活中我们通常使用彩色图像来记录事物。然而在很多场景中，灰度图像依然有着非常广泛的应用。例如黑白印刷由于比彩色印刷更廉价，依然受到大众欢迎。在打印中通常需要将彩色图像转为灰度图像，并对图像的灰度进行增强以提高打印品质。另外，在图像的边缘检测、特征提取、图像分割及图像分类等图像处理过程中也经常使用灰度图像，因为相比于彩色图像，灰度图像数据量小，便于存储和提高运算效率。因此使用这些算法进行图像处理之前，经常需要预先将彩色图像转换为灰度图像。

记一幅像素大小为$m\times n$的真彩色图像的红色、绿色和蓝色分量矩阵分别为$\boldsymbol{R}_{m\times n}$, $\boldsymbol{G}_{m\times n}$和$\boldsymbol{B}_{m\times n}$，一个简单但受欢迎的真彩色图像灰度化公式[35]为

$$\boldsymbol{I}_{m\times n}=\alpha_r\boldsymbol{R}_{m\times n}+\alpha_g\boldsymbol{G}_{m\times n}+\alpha_b\boldsymbol{B}_{m\times n} \tag{7}$$

其中，$\boldsymbol{I}_{m\times n}$为灰度图像的像素矩阵，$\alpha_r$, α_g和α_b为非负系数，且满足

$$\alpha_r+\alpha_g+\alpha_b=1 \tag{8}$$

系数$(\alpha_r,\alpha_g,\alpha_b)$可以根据需求及灰度效果选择不同的组合，其中$(\alpha_r,\alpha_g,\alpha_b)=(0.3,0.59,0.11)$[36]在使用中非常流行，即

$$\boldsymbol{I}_{m\times n}=0.3\times\boldsymbol{R}_{m\times n}+0.59\times\boldsymbol{G}_{m\times n}+0.11\times\boldsymbol{B}_{m\times n} \tag{9}$$

欲通过DM-IMCA的存内计算特性来实现式(9)，需对公式进行定点量化及转换等预处理。对式(9)进行位宽量化(放大2^{16}倍)并采取去尾取整，可得

$$\begin{aligned}\boldsymbol{I}_{m\times n}=&(19595\times\boldsymbol{R}_{m\times n}+38469\times\boldsymbol{G}_{m\times n}\\&+7472\times\boldsymbol{B}_{m\times n})\gg 16\end{aligned} \tag{10}$$

进一步将其精度缩减至3位和2位，分别可得

$$\boldsymbol{I}_{m\times n}=(1\times\boldsymbol{R}_{m\times n}+2\times\boldsymbol{G}_{m\times n}+1\times\boldsymbol{B}_{m\times n})\gg 2 \tag{11}$$

和

$$\boldsymbol{I}_{m\times n}=(2\times\boldsymbol{R}_{m\times n}+5\times\boldsymbol{G}_{m\times n}+1\times\boldsymbol{B}_{m\times n})\gg 3 \tag{12}$$

因此，通过位宽量化及精度缩减，可以将图像灰度化公式变换成只包含加法和移位运算。例如式(11)中的$2\times\boldsymbol{G}_{m\times n}$可以通过DM-IMCA的一次加法(madd)或者移位(msrl)实现，式(12)中的$5\times\boldsymbol{G}_{m\times n}$可以通过DM-IMCA的两次存内1 bit移位及一次加法实现。对于一个像素点来说，若使用式(11)来将RGB值转换为灰度值，需要通过4次存内运算来完成。一般图像由比较大的像素点阵(如16×16, 64×64或256×256)组成，图像灰度化运算量比较大。但由于DM-IMCA支持自动化及自适应的存内向量计算，仍然可以对图像灰度化运算提供可观的加速比。对一幅像素为28×28的RGB图像进行灰度化，实验结果表明，对于相同的量化后算法，相比于基准系统，在DM-IMCA中实现可以获得约10倍的运算加速比。

7 结束语

本文介绍了基于SRAM进行存内逻辑计算的原理，以及基于SRAM的通用存算一体架构平台DM-IMCA。随后，本文选取了OTP加密、哈希算法、二值深度神经网络以及图像灰度处理等几个轻量级物联网应用，对其进行算法分析及拆分，将数据密集计算部分算法映射到一款通用型存算一体架构平台DM-IMCA的计算型SRAM中，以达到对应用进行加速和提高能效的目的。这些实例表明DM-IMCA在物联网领域具有较高的应用价值，且为后续的应用提供了思路与方法。

参 考 文 献

[1] JIN Xiaolong, WAH B W, CHENG Xueqi, *et al*. Significance and challenges of big data research[J]. *Big Data Research*, 2015, 2(2): 59–64. doi: 10.1016/j.bdr.2015.01.006.

[2] SCHILIZZI R T. The square kilometer array[C]. Proceedings SHE, Ground-based Telescopes, Glasgow, UK, 2004: 62–71.

[3] JONGERIUS R, WIJNHOLDS S, NIJBOER R, *et al*. An end-to-end computing model for the square kilometre array[J]. *Computer*, 2014, 47(9): 48–54. doi: 10.1109/MC.2014.235.

[4] ZASLAVSKY A, PERERA C, and GEORGAKOPOULOS D. Sensing as a service and big data[J]. arXiv preprint arXiv: 1301.0159, 2013.

[5] FLORIDI L. Big data and their epistemological challenge[J]. *Philosophy & Technology*, 2012, 25(4): 435–437. doi: 10.1007/s13347-012-0093-4.

[6] REINSEL D, GANTZ J, and RYDNING J. Data age 2025: The digitization of the world from edge to core[R]. IDC White Paper–#US44413318, 2018.

[7] SIEGL P, BUCHTY R, and BEREKOVIC M. Data-Centric computing frontiers: A survey on processing-in-memory[C]. Proceedings of the Second International Symposium on Memory Systems (MEMSYS'16), Alexandria, USA, 2016: 295–308. doi: 10.1145/2989081.2989087.

[8] PATTERSON D, ANDERSON T, CARDWELL N, *et al*. A case for intelligent RAM[J]. *IEEE Micro*, 1997, 17(2): 34–44. doi: 10.1109/40.592312.

[9] ROGERS B M, KRISHNA A, BELL G B, *et al*. Scaling the bandwidth wall: Challenges in and avenues for CMP scaling[J]. *ACM SIGARCH Computer Architecture News*, 2009, 37(3): 371–382. doi: 10.1145/1555815.1555801.

[10] DAS R. Blurring the lines between memory and computation[J]. *IEEE Micro*, 2017, 37(6): 13–15. doi: 10.1109/MM.2017.4241340.

[11] ZHU Qiuling, AKIN B, SUMBUL H E, *et al*. A 3D-stacked logic-in-memory accelerator for application-specific data intensive computing[C]. 2013 IEEE International 3D Systems Integration Conference (3DIC), San Francisco, USA, 2013: 1–7. doi: 10.1109/3DIC.2013.6702348.

[12] AHN J, HONG S, YOO S, *et al*. A scalable processing-in-memory accelerator for parallel graph processing[C]. Proceeding of the ACM/IEEE 42nd Annual International

Symposium on Computer Architecture (ISCA'15), Portland, USA, 2015: 105–117. doi: 10.1145/2749469.2750386.

[13] AHN J, YOO S, MUTLU O, *et al*. PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture[C]. The 42nd Annual International Symposium on Computer Architecture (ISCA'15), Portland, USA, 2015: 336–348. doi: 10.1145/2749469.2750385.

[14] CHI Ping, LI Shuangchen, XU Cong, *et al*. PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory[J]. *ACM SIGARCH Computer Architecture News*, 2016, 44(3): 27–39. doi: 10.1145/3007787.3001140.

[15] HALAWANI Y, MOHAMMAD B, LEBDEH M A, *et al*. ReRAM-based in-memory computing for search engine and neural network applications[J]. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2019, 9(2): 388–397. doi: 10.1109/JETCAS.2019.2909317.

[16] ZHANG Bin, CHEN Weilin, ZENG Jianmin, *et al*. 90% yield production of polymer nano-memristor for in-memory computing[J]. *Nature Communications*, 2021, 12(1): 1984. doi: 10.1038/s41467-021-22243-8.

[17] JELOKA S, AKESH N B, SYLVESTER D, *et al*. A 28 nm configurable memory (TCAM/BCAM/SRAM) using push-rule 6T bit cell enabling logic-in-memory[J]. *IEEE Journal of Solid-State Circuits*, 2016, 51(4): 1009–1021. doi: 10.1109/JSSC.2016.2515510.

[18] AGA S, JELOKA S, SUBRAMANIYAN A, *et al*. Compute caches[C]. 2017 IEEE International Symposium on High Performance Computer Architecture, Austin, USA, 2017: 481–492. doi: 10.1109/hpca.2017.21.

[19] ZHANG Yiqun, XU Li, DONG Qing, *et al*. Recryptor: A reconfigurable cryptographic cortex-M0 processor with in-memory and near-memory computing for IoT security[J]. *IEEE Journal of Solid-State Circuits*, 2018, 53(4): 995–1005. doi: 10.1109/JSSC.2017.2776302.

[20] ZENG Jianmin, ZHANG Zhang, CHEN Runhao, *et al*. DM-IMCA: A dual-mode in-memory computing architecture for general purpose processing[J]. *IEICE Electronics Express*, 2020, 17(4): 20200005. doi: 10.1587/elex.17.20200005.

[21] AGRAWAL A, JAISWAL A, ROY D, *et al*. Xcel-RAM: Accelerating binary neural networks in high-throughput SRAM compute arrays[J]. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2019, 66(8): 3064–3076. doi: 10.1109/TCSI.2019.2907488.

[22] YIN Shihui, JIANG Zhewei, SEO J S, *et al*. XNOR-SRAM: In-memory computing SRAM macro for binary/ternary deep neural networks[J]. *IEEE Journal of Solid-State Circuits*, 2020, 55(6): 1733–1743. doi: 10.1109/jssc.2019.2963616.

[23] LIU Rui, PENG Xiaochen, SUN Xiaoyu, *et al*. Parallelizing SRAM arrays with customized bit-cell for binary neural networks[C]. 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), San Francisco, USA, 2018: 1–6. doi: 10.1109/DAC.2018.8465935.

[24] KANG Mingu, GONUGONDLA S K, PATIL A, *et al*. A multi-functional in-memory inference processor using a standard 6T SRAM array[J]. *IEEE Journal of Solid-State Circuits*, 2018, 53(2): 642–655. doi: 10.1109/JSSC.2017.2782087.

[25] BELLOVIN S M. Frank miller: Inventor of the one-time pad[J]. *Cryptologia*, 2011, 35(3): 203–222. doi: 10.1080/01611194.2011.583711.

[26] SOBTI R and GEETHA G. Cryptographic hash functions: A review[J]. *IJCSI International Journal of Computer Science Issues*, 2012, 9(2): 461–479.

[27] SEOK B, PARK J, and PARK J H. A lightweight hash-based blockchain architecture for industrial IoT[J]. *Applied Sciences*, 2019, 9(18): 3740. doi: 10.3390/app9183740.

[28] SZE V, CHEN Y H, YANG T J, *et al*. Efficient processing of deep neural networks: A tutorial and survey[J]. *Proceedings of the IEEE*, 2017, 105(12): 2295–2329. doi: 10.1109/JPROC.2017.2761740.

[29] LECUN Y, BOTTOU L, BENGIO Y, *et al*. Gradient-based learning applied to document recognition[J]. *Proceedings of the IEEE*, 1998, 86(11): 2278–2324. doi: 10.1109/5.726791.

[30] SIMONYAN K and ZISSERMAN A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv: 1409.1556, 2014.

[31] HE Kaiming, ZHANG Xiangyu, REN Shaoqing, *et al*. Deep residual learning for image recognition[C]. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16), Las Vegas, USA, 2016: 770–778. doi: 10.1109/CVPR.2016.90.

[32] COURBARIAUX M, HUBARA I, SOUDRY D, *et al*. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or –1[J]. arXiv preprint arXiv: 1602.02830, 2016.

[33] RASTEGARI M, ORDONEZ V, REDMON J, *et al*. XNOR-Net: ImageNet classification using binary convolutional neural networks[C]. The 14th European Conference on Computer Vision, Amsterdam, The Netherlands, 2016: 525–542. doi: 10.1007/978-3-319-46493-0_32.

[34] FOLEY J D, VAN F D, VAN DAM A, *et al*. Computer Graphics: Principles and Practice[M]. Boston: Addison-Wesley Professional, 1996: 1–1175.

[35] WAN Yi and XIE Qisong. A novel framework for optimal RGB to grayscale image conversion[C]. The 2016 8th

International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC), Hangzhou, China, 2016: 345–348. doi: 10.1109/IHMSC.2016.201.

[36] PRATT W K. Introduction to Digital Image Processing[M]. Boca Ration: CRC Press, 2013: 1–756.

曾剑敏：男，1991年生，博士生，研究方向为存算一体架构设计.

张 章：男，1982年生，教授，主要研究方向为集成电路设计及基于混合器件的AI神经形态芯片设计.

虞志益：男，1977年生，教授，研究方向为处理器架构设计、基于非易失器件、面向人工智能深度学习等应用的电路与系统设计.

解光军：男，1970年生，教授，主要研究方向为集成电路及新型器件电路设计.

责任编辑：马秀强