计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2019.20180157
56(4):798-809, 2019

一种基于斯格明子介质的高效存内计算框架

刘必成　顾海峰　陈铭松　谷守珍　陈闻杰
(上海市高可信计算重点实验室(华东师范大学)　上海　200062)
(51151500030@stu.ecnu.edu.cn)

An Efficient Processing In Memory Framework Based on Skyrmion Material

Liu Bicheng, Gu Haifeng, Chen Mingsong, Gu Shouzhen, and Chen Wenjie
(*Shanghai Key Laboratory of Trustworthy Computing* (*East China Normal University*), *Shanghai* 200062)

Abstract　As a new computing paradigm, processing in memory (PIM) allows the parallel computation in both processors and memories, which drastically reduce the movements between computation units and storage units. Therefore, PIM can be considered as an efficient technology to somewhat address the shortcomings of the von neumann architecture. Compared with traditional random access memories, racetrack memory has many merits including high density, non-volatility, and low static power. Therefore, it can be used for efficient PIM computing. To address the shortages of domain-wall based PIM, this paper proposes a novel PIM framework based on the Skyrmion material. In this framework, we use Skyrmion-based racetrack memories to construct storage units, and use Skyrmion-based logic gates to compose both adders and multipliers for the computation units. Since our framework does not need CMOS (complementary metal oxide semiconductor) circuits to assist the underlying computation unit construction, the design complexity is significantly reduced. Meanwhile, based on our proposed optimization methods for read and write operations at the circuit layer and address mapping mode of the memory at the system level, the performance of our framework is drastically improved. Experimental results show that compared with domain-wall based PIM framework, our approach can achieve 48.1% time improvement and 42.9% energy savings on average.

Key words　Skyrmion; non-volatile memory; processing in memory (PIM); racetrack memory; address mapping

摘　要　存内计算(processing in memory, PIM)作为一种新兴的技术,支持数据在存储单元内就地处理,减少了数据的移动并增加了数据的并行处理,在一定程度上弥补了冯·诺依曼架构的缺陷.和传统易失随机存储介质相比,赛道型内存(racetrack memory, RM)具有密度大、非易失且静态功耗低等特点,支持高效的存内计算.为解决性能与功耗问题,提出了一种新型的基于斯格明子(Skyrmion)介质的非易失性存内计算框架.该框架采用斯格明子赛道内存(Skyrmion-based racetrack memory)作为存储

收稿日期:2018-03-07;**修回日期**:2018-09-20
基金项目:国家自然科学基金项目(61872147,61702187);"核高基"国家科技重大专项基金项目(2017Z01038102-002);中央高校基本科研业务费专项资金项目;上海市自然科学基金项目(15ZR1410000)
This work was supported by the National Natural Science Foundation of China (61872147, 61702187), the National Science and Technology Major Project of Hegaoji (2017Z01038102-002), the Fundamental Research Funds for the Central Universities, and the Natural Science Foundation of Shanghai (15ZR1410000).
通信作者:陈铭松(mschen@sei.ecnu.edu.cn)

单元,采用斯格明子逻辑门(Skyrmion-based logic gate)构成的加法/乘法器组成计算单元,无须大量CMOS(complementary metal oxide semiconductor)电路辅助,设计复杂度大大降低.同时,通过在电路级优化存储单元读写端口数目与在系统级改进内存地址映射方式,大幅提高该框架的运行效率.实验结果表明:相比基于磁畴壁(domain-wall)的非易失性存内计算框架,提出的框架在运行时间上节省了48.1%,同时在能耗上节省了42.9%.

关键词 斯格明子;非易失性存储;存内计算;赛道存储;地址映射

中图法分类号 TP333

当今世界已进入大数据时代,各种现代应用对数据处理速度的要求越来越高.然而在传统的冯·诺依曼架构中,数据的存储和处理各自分离,同时数据量与处理速度之间的差距也在逐步拉大,严重制约了系统效率的进一步提高.为了克服这个困难,文献[1-2]提出了新型存内计算(processing in memory, PIM)架构,并受到广泛关注和研究[3-4].在存内计算架构中,存储单元和计算单元在内存中紧密地结合在一起,使得数据可以直接在内存中就地进行处理,从而极大地减少了数据在内存和处理器之间的频繁移动且增加了数据处理的并行性.

虽然存内计算架构在一定程度上缓解了"数据搬运"的瓶颈问题,然而由于传统存内计算建立在易失性存储器介质之上,其物理特性限制导致整个系统泄漏功耗和动态功耗随着处理数据量的增加而急剧增长.近期各种新型非易失性内存介质(non-volatile memory, NVM)正因其区别与传统介质的低漏电率、高密度等一系列优良的特性而受到广泛关注[5-7].典型的包括相变存储器(phase change memory, PCRAM)、自旋力矩存储器(spin-transfer torque memory, STT-RAM)、赛道型存储器(racetrack memory, RM)等.其中RM通过将多个比特的数据存储在一条类似磁带的纳米线上,提供了比自旋力矩存储器更高的存储密度,比相变存储器更高的写入寿命,以及接近静态随机存取存储器(static random access memory, SRAM)的读写速度[8-10].

赛道型存储的本身物理结构决定了其不但适用于存储数据,也非常容易组成各种逻辑结构来进行数据处理,因此可以用来作为存内计算的介质.第1代赛道型存储器是基于磁畴壁(domain-wall)介质的,文献[11]在此基础上提出了一种较为通用的存内计算架构.然而这种基于磁畴壁介质的存内架构依然需要大量的CMOS(complementary metal oxide semiconductor)外围电路来进行辅助计算,导致了计算单元体积和能耗的增加.

最近新型的基于斯格明子介质的第2代赛道型存储器被提出[12-15].相比磁畴壁介质,斯格明子介质具有密度更高、能耗更低、稳定性更强以及更少受限于材料等一系列优良特性,非常适合作为下一代存内计算的介质.同样这种介质特性也非常适合使用在嵌入式系统中,甚至可以用来构建基于嵌入式系统的移动存内计算框架.然而目前对于斯格明子介质的研究主要集中于硬件存储功能,缺乏关于计算功能的研究,系统层次以及具体应用实现也很少涉及[16].另一方面由于斯格明子-赛道型存储器特有的条带状物理结构,使其具有特有的顺序读写特性,如何用其替代现有存内计算架构下的存储单元也是亟待解决的问题.

针对以上问题,本文提出了一种基于斯格明子介质的存内计算框架,主要贡献有4点:1)结合斯格明子介质本身的物理特性,由斯格明子逻辑门组成加法器、乘法器等计算单元并进行优化,极大地减少了CMOS辅助电路的使用,提高了计算效率;2)在硬件电路层面上对于基本存储单元读写端口数等参数进行探讨,并通过实验优化配置;3)在系统层上对内存的地址映射方式进行改进,提高了整个系统的运行效率;4)以通用的图像锐化程序为例详细说明了程序在内存框架中的工作流程,同时将本文提出的基于斯格明子介质的内存框架与目前最先进的基于磁畴壁的存内计算框架进行实验对比.

1 基于斯格明子介质的存内计算框架

基于斯格明子介质的存内计算主要包含2部分:基于斯格明子介质的存储单元和计算单元,其中存储单元即斯格明子-赛道型存储器,是整个框架的基础.

1.1 斯格明子-赛道型存储器件

斯格明子-赛道型存储器[12-13],区别于磁畴壁-赛道型存储器,是一种基于斯格明子编码的非易失性

存储器.如图 1 所示,数据通过斯格明子编码之后存储在一条单一的铁磁纳米线(nanowire)器件上.纳米线上的斯格明子由电压控制的磁各向异性(voltage-controlled magnetic anisotropy, VCMA)门所隔离,每 2 个门之间存储一位数据.如果此区间内存在斯格明子则代表数据 1,如果不存在斯格明子则代表数据 0.斯格明子-赛道型存储器件有 3 项基本操作:移位、读和写,其中具有移位操作是其最重要的特性.

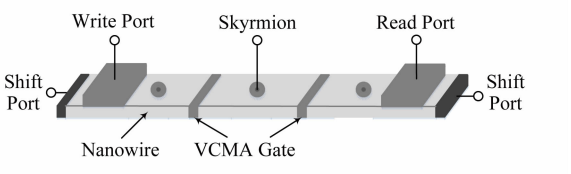


Fig. 1 Skyrmion based nanowire device

图 1 斯格明子-赛道型存储器件结构图

斯格明子-赛道型存储器件的移位操作,是指在磁各向异性门打开时纳米线上的斯格明子可以通过在存储器两端移位端口(shift port)施加电流来进行向左或向右移动.为了保证移位操作之后记录在纳米线上的数据不丢失,在纳米线两端应当有冗余的存储位供位移操作使用.整体来说所有比特数据的移位都类似于磁带操作,和移位寄存器类似.

斯格明子-赛道型存储器件读、写操作的基本原理类似.在存储器器件中有读写端口(write/read port),即沿着纳米线方向放置的一个强磁化铁磁层,但是其和纳米线之间由较薄的绝缘层隔开.这样的三明治结构形成了磁隧道结(magnetic tunnel junctions, MTJs).通过向写端口的 MTJ 结构中注入自旋极化电流(spin-transfer current)就可以在纳米线上产生一个斯格明子.同样读端口也是一个 MTJ 结构,通过检测读端口 MTJ 隧穿电导(tunneling conductance)的变化就可以得知纳米线上当前位置是否存在斯格明子,即数据是 0 还是 1.需要注意的是,由于写和读操作只能在固定的 MTJ 端口处进行,因此纳米线上比特位数据的操作需要移动到与 MTJ 固定层对齐的位置才能进行,而移位操作的方向和速度取决于控制电流的方向和幅度.

1.2 基于斯格明子介质的存内计算框架

传统上所有的数据都是保存在和处理器分离的主存中,二者通过总线相连接.因此在程序执行过程中所有的数据都需要迁移到处理器中,并在处理完成之后再次写回.对于以数据为导向的应用,这将产生严重的通信堵塞,从而大大降低总体性能.此外在传统的内存中保存大量的数据也将产生明显的待机能耗.

为了克服上述 2 个问题,我们使用基于非易失性内存的计算架构.首先存内计算架构在一定程度上解决了数据传输瓶颈的问题,也减少了数据传输的能耗;其次非易失性内存在极大地减少待机功耗的同时也降低了内存的动态功耗.基于斯格明子-赛道型存储器的存内计算平台整体结构如图 2 所示,其中存内计算单元与存储单元以分布式的方式组合成存储-计算单元组,这样许多频繁处理数据的操作可以在内存内部完成而无需与外部处理器进行通信,从而极大地节省了时间与能耗的开销.同时分布式的内存处理单元也可以提供巨大的线程级并行性,从而极大地提高系统吞吐量.

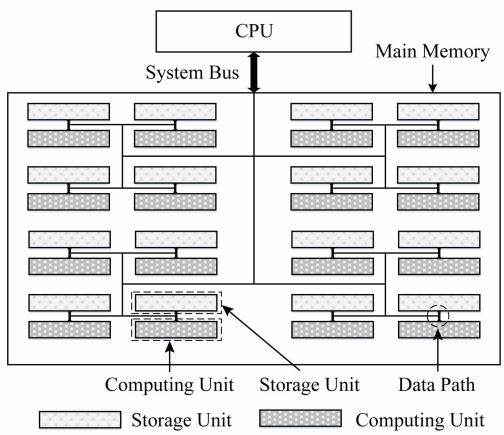


Fig. 2 The structure of PIM platform

图 2 存内计算架构

在本文提出的基于斯格明子的存内计算框架中,内存存储单元由基于斯格明子的赛道型存储器构成,从而受益于其低漏电功耗、非易失性以及稳健性等优点.同时存内计算单元纯粹由基于斯格明子逻辑门的加法器、乘法器等组成,只需要极少的 CMOS 电路辅助,因此总体漏电功耗和处理数据所需的动态功耗和时间消耗都极大地减少.在本文提出的存内计算框架中,存储-计算单元组之间通过 H 型内部数据通路相连接,这样单元组与单元组之间的数据可以随时根据需要进行传输,而外部处理器(即 CPU)主要负责将控制指令传输给内存内部的控制单元,由内部控制单元负责内存中存储与计算单元具体数据的调度处理.由于斯格明子既具有计算功能又具有存储功能,因此在本文提出的存内计算框架中,计算单元得到的结果将直接写入存储单元中,即存储单元本身完成了类似寄存器的时序逻辑功能.

2 基于斯格明子介质的计算单元设计

本节首先从硬件层面考虑,提出基于斯格明子逻辑门的加法逻辑单元和进位逻辑单元设计,再进一步提出整个全加器的设计,最后在全加器设计的基础上提出了基于斯格明子逻辑门乘法器的设计,并进一步对加法器进行了优化.

2.1 基于斯格明子逻辑门的加法逻辑

典型的逻辑和运算由 2 个异或门组成,然而异或逻辑门无法直接使用斯格明子器件实现.这个问题可以通过斯格明子逻辑门组合来实现[14-15].其中文献[14]实现了基于斯格明子的逻辑与门和逻辑或门,同时包含基于斯格明子的复制(duplication)逻辑,而文献[15]中实现了基于斯格明子的逻辑与非门和逻辑或非门.在此基础上本文构建了基于斯格明子的异或逻辑门.如图 3(b)所示,基于斯格明子的异或逻辑门由 1 个或门,1 个与非门以及 1 个与门组成.需要注意的是,图 3(b)中 OR2-Gate 是与非门 NAND-Gate 的一部分,输入部分 A_n 和 B_n 分别代表数据 A 和 B 的第 n 位.正如图 3(a)所示,A 和 B 是一个存储在斯格明子纳米线上 8 b 的数据.在斯格明子纳米线上,如果某个位置存在有斯格明子,它就代表数值 1;如果没有斯格明子,它就表示数值 0.因此图 3(a)以二进制形式表示 $A=10111001$,$B=10101110$.

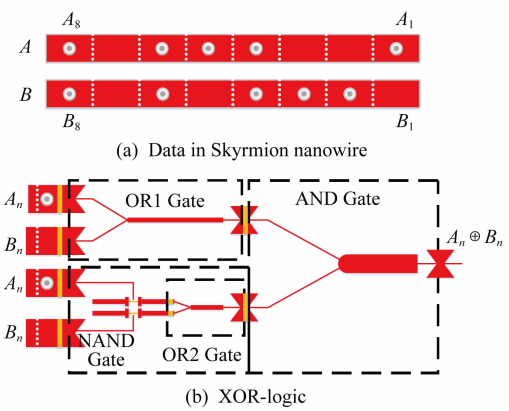


Fig. 3 Skyrmion nanowire-based XOR-logic

图 3 基于斯格明子的异或逻辑单元

当 $n=1$ 时异或逻辑单元工作步骤有 3 个:

1) 操作数 A_1 和 B_1 同时进入逻辑门 OR1 和 NAND.由于 $A_1=1$,$B_1=0$ 也即有一个斯格明子进入逻辑门 OR1,一个斯格明子进入逻辑门 NAND.

2) 代表 A_1 的斯格明子分别通过逻辑门 OR1 和 NAND 并保持不变.

3) 从逻辑门 OR1 和 NAND 出来的 2 个斯格明子同时进入逻辑门 AND,最终合并成一个斯格明子,从而可以得到 $A_1 \oplus B_1=1$.

通过基于斯格明子纳米线器件的异或逻辑单元我们可以实现带进位的加法逻辑单元($SUM=A_n \oplus B_n \oplus C_{in}$,其中 C_{in} 为进位).即通过组合 2 个异或逻辑单元:第 1 个异或逻辑单元输入是 A_n 和 B_n,输出是 $A_n \oplus B_n$;第 2 异或逻辑单元输入是 $A_n \oplus B_n$ 和 C_{in},而输出是当前位的进位和 SUM.

2.2 基于斯格明子逻辑门的进位逻辑

一个典型的进位逻辑由 3 个与门和 2 个或门组成.图 4 显示了基于斯格明子逻辑门的进位逻辑单元具体设计细节.

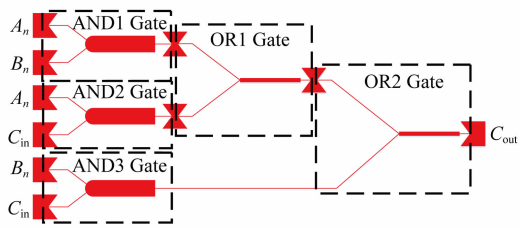


Fig. 4 Skyrmion nanowire-based carry-logic

图 4 基于斯格明子的进位逻辑门

如图 4 所示,进位逻辑单元有 3 组输入:A_n 和 B_n,A_n 和 C_{in},B_n 和 C_{in},以及一个输出:C_{out}.其中输入 C_{in} 为第 $n-1$ 位的进位,输出 C_{out} 为第 n 位的进位.进位逻辑单元具体实现细节有 4 点:

1) 代表上述 3 组输入第 n 位数值的斯格明子粒子分别进入了 3 个与门,即 AND1~AND3.

2) 第 1 个与门 AND1 的输出和第 2 个与门 AND2 的输出将同时进入第 1 个或门 OR1.第 3 个与门 AND3 的输出将在进入第 2 个或门 OR2 之前等待 OR1 的输出.

3) 第 1 个或门 OR1 的输出与第 3 个与门 AND3 的输出同时进入第 2 个或门 OR1.

4) 第 2 个或门的输出即进位的值 C_{out}.

2.3 基于斯格明子逻辑门的全加器

基于斯格明子逻辑门的全加器如图 5 所示,此全加器由 3 个主要部分构成:第 1 部分(PART1)是和运算部分,由第 1 个异或逻辑组成,它的输入是 A_n 和 B_n,输出是 $A_n \oplus B_n$.同时第 1 部分还有一个复制逻辑(duplication)以便为第 3 部分(PART3)提供输入.第 2 部分(PART2)是进位逻辑,其输入是

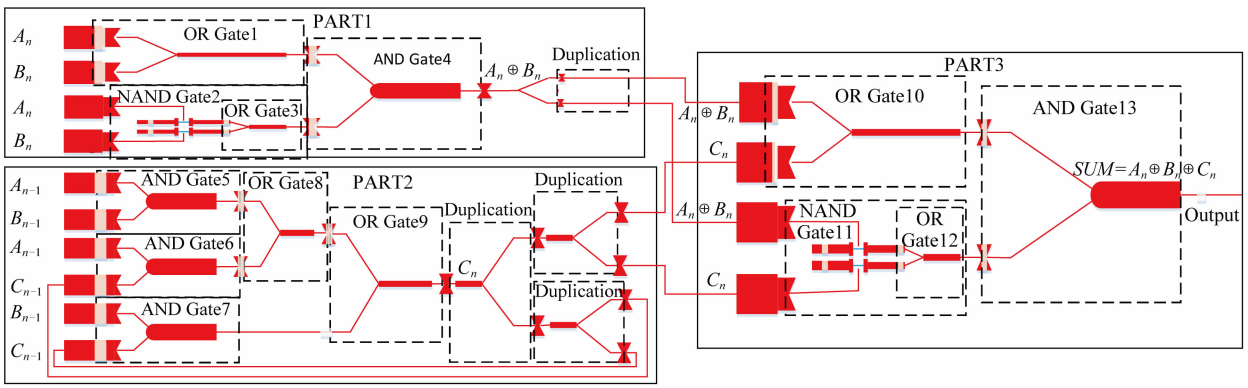


Fig. 5 Skyrmion nanowire-based full adder

图 5 基于斯格明子逻辑门的全加器

A_{n-1}, B_{n-1}, C_{n-1}, 而输出是 C_n 即 $n-1$ 位的进位. 同时 C_n 会复制 4 份, 其中 2 份作为第 3 部分(PART3)的输入, 另外 2 份作为下一位加法的输入. 第 3 部分(PART3)由第 2 个异或逻辑构成, 其输入是 $A_n \oplus B_n$ 和 C_n, 即第 1 部分和第 2 部分的输出, 而输出就是全加器的最终结果: $SUM = A_n \oplus B_n \oplus C_n$.

图 5 中的针孔形状部分代表一种能量势垒(energy barrier)[15], 这种能量势垒在电压为正的时候可以阻止斯格明子通过, 而在电压为 0 的时候允许斯格明子通过, 从而起到类似开关的作用. 通过能量势垒的开关可以使得斯格明子同步进入逻辑门的 2 个输入端以保证逻辑门的正常工作. 注意, 当 $n=1$ 时, A_n 代表数据 A 的第 1 位, 此时 A_{n-1} 不存在即无任何输入, B_{n-1} 与 C_n 也相同.

不同斯格明子逻辑门的传播时延已在文献[14-15]中给出. 基于已知的各种逻辑门的工作时间, 当整个系统的工作频率为 1 000 MHz 时, 通过计算得知全加器进行 1 位的加法需要 11 个时钟周期. 考虑到能量势垒开关在使逻辑门输入同步的同时也使得各个逻辑门之间相互隔离, 因此当进行多个位的加法时可以利用此特性对全加器的工作流程进一步优化. 受 CPU 流水线优化技术启发, 我们对全加器进行了优化: 在 1 位加法计算完成之前就允许下一位的数据进入全加器, 从而极大地提高了整体工作效率.

经过优化后的全加器电路时序图如图 6 所示. 其中横坐标的数字 1～19 分别代表 19 个时钟周期, 每个时钟周期为 1 ns; 纵坐标的 Gate 1～13 分别对应图 6 所示全加器中对应的 13 个逻辑门的控制电压, 即每个逻辑门输入端口处能量势垒开关的电压. 经过优化后的全加器主要时序逻辑为

1) 第 1 个时钟周期. Gate1～2, Gate5～7 对应的控制电压为低电压, 因此对应输入 端口的斯格明子(也即 A_n, B_n, A_{n-1}, B_{n-1}, C_{n-1})可以进入 OR-Gate1, NAND-Gate2 和 AND-Gate5. 为了保证输入同步, Gate3～4, Gate8～9 和 Gate11 对应的控制电压为高电压. 其他逻辑门对应的控制电压均保持低电压, 因为这些逻辑门还未被使用.

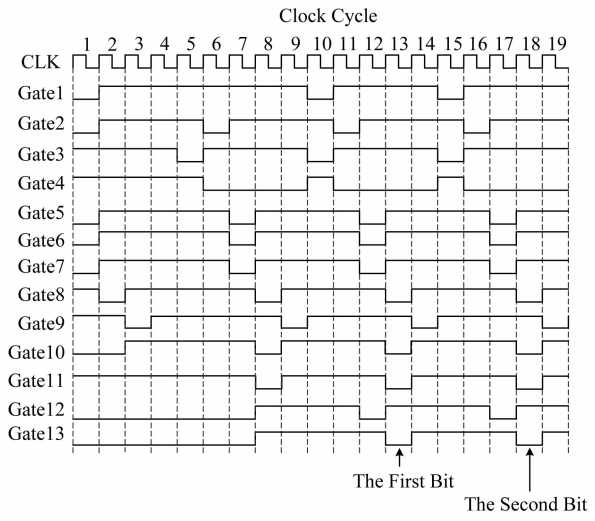


Fig. 6 Timing diagram of 8-bit full adder

图 6 全加器电压控制时序图

2) 第 2 个时钟周期. Gate1～2, Gate5～7 对应的控制电压变为高电压以阻止斯格明子进入对应逻辑门, 同时 Gate3～4, Gate9 和 Gate11 的控制电压继续保持高电压以完成逻辑门同步功能. Gate8 的控制电压从高电压转为低电压从而使得斯格明子进入 OR-Gate8, 其他逻辑门的对应控制电压依然保持不变.

3) 第 3 个时钟周期. Gate9 的控制电压转为低电压以便斯格明子进入 OR-Gate9, 同时后续 Gate10 的控制电压转变为高电压以完成同步功能, 其他逻辑门的控制电压保持不变.

4) 第 6 个时钟周期. Gate4 的控制电压从高电压变为低电压以便斯格明子进入 AND-Gate4;同时 Gate2 的控制电压变为低电压以便允许下个比特的数据进入 NAND-Gate2,后续 Gate3 依然保持高电压.

5) 第 8 个时钟周期. Gate10～11 的控制电压转为低电压从而 $A_n \oplus B_n$ 和 C_n 对应的斯格明子可以进入 OR-Gate10 和 NAND-Gate11. Gate12～13 的控制电压为了保持同步应变为高电压状态. 同时 Gate5～7 的控住电压转为高电压,而 Gate8 的控制电压转为低电压.

6) 第 13 个时钟周期. Gate5～7 的控制电压转为高电压以阻止斯格明子进入逻辑门,同时 Gate8, Gate10, Gate11, Gate13 的控制电压转为低电压.

7) 第 14 个时钟周期. 可以通过输出端口是否有斯格明子判断 SUM 的值是 0 还是 1,从而得到求和运算的第 1 个位数值.

8) 第 15 个时钟周期及以后. 不断重复第 10～14 个时钟周期的状态,每隔 5 个时钟周期就可以读出和的下一位数值.

如图 6 所示,经过计算可以得知第 1 位的加法需要 14 个时钟周期(每个时钟周期 1 ns),而从第 2 位开始每 5 个时钟周期全加器就可以完成一个位的加法. 这是由于经过优化后全加器内部各个逻辑门之间相互独立运行,从而可以获得类似流水线的优化效果,考虑到在进行大量数据处理时或者随着运行频率的进一步提高优化效果依然可以进一步提高. 对于常用的 8 b 的加法,本文提出的基于斯格明子介质的全加器经过优化后只需要 49 个时钟周期即 49 ns,相比基于磁畴壁的第 1 代赛道存储内存加法器 (8 位加法需要 108 ns)[1] 快了 2.2 倍.

2.4 基于斯格明子逻辑门的乘法器

通常来说乘法可以分解为多次移位操作和加法操作,而本文提出的存内计算框架中全加器可以通过纯粹的基于斯格明子逻辑门实现,移位操作又是斯格明子纳米线器件自带的能力,因此本文提出如图 7 所示基于斯格明子逻辑门的 8 位乘法器.

在图 7 中,A_n 代表当对应操作数 B 的第 n 位为 1 时,需要将操作数 A 左移 $n-1$ 位. 例如当操作数 A 和 B 的二进制形式分别为 1101 和 111 时,有 $A_0=1101$, $A_1=11010$, $A_2=110100$,此时 A 乘以 B 就等于 $A_0+A_1+A_2$. 由于操作数 A 存储在斯格明子纳米线上,而纳米线器件本身就支持移位操作,因此 A_n 可以通过将操作数 A 左移 n 位得到,再直接输入全加器中得到乘法结果. 因此基于斯格明子的乘法器可以通过重复利用已有的斯格明子全加器和本身的移位来实现,因此大大减少了计算逻辑单元所需的空间以及时间,同时也减少了实现存内计算框架的复杂程度.

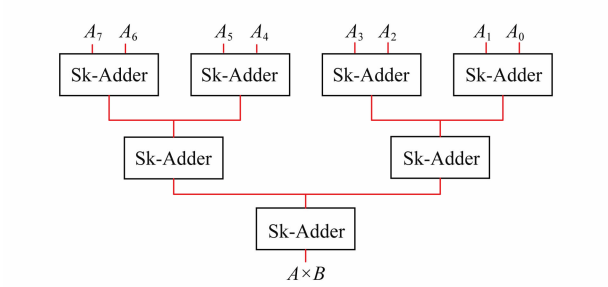


Fig. 7 8-bit Skyrmion nanowire-based multiplier

图 7 基于斯格明子的 8 位乘法器

3 基于斯格明子介质的存储单元设计

在存内计算框架中存储单元与计算单元一同对整个系统的性能起着至关重要的作用. 而斯格明子-赛道型存储器本身的物理特性决定其与传统的 DRAM 存储器随机读写的方式并不相同,斯格明子-赛道型存储器具有顺序读写的特性. 因此我们无法简单地用斯格明子存储单元直接替代 DRAM 存储单元. 为了进一步提高斯格明子存内计算框架的效率,我们需要根据斯格明子-赛道型存储器的本身物理特性来从底层硬件及系统软件 2 个层面考虑存内计算框架中存储单元的设计.

3.1 斯格明子存储单元底层硬件设计

基于斯格明子的基本存储单元具体结构如图 8 所示. 其中存储部分由 RT0 到 RT3 共 4 条基本赛道组成. 每条赛道上可能有 n 个读写端口(图 8 中圆

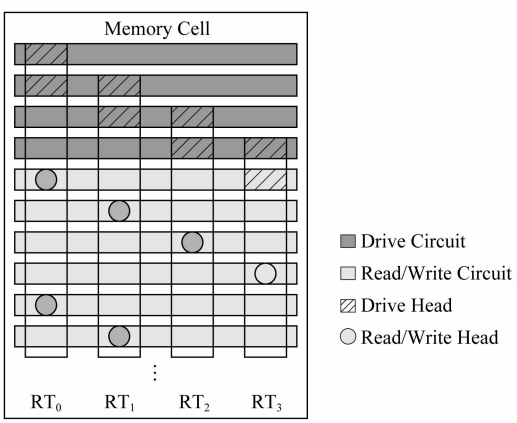


Fig. 8 Skyrmion based memory cell

图 8 基于斯格明子介质的存储单元

形部分),这样每个单元可以一次读写 $4n$(单位为 b).典型的赛道型存储器具有 3 个基本操作即读、写以及移位.由于其中移动数据的移位操作占据绝大部分的时间和能耗,所以如何在不影响系统性能的情况下尽量减少数据的移位操作是亟待解决的问题.

在图 8 所示结构中,减少移位最直观有效的方法是增加读写端口的数量.但是由于读写端口本身会占用大量的空间,因此增加读写端口会相应降低存储的密度,同时会带来读写延时、能耗的增加以及实现工艺的复杂化,因此需要在增加读写端口与减少数据移位之间寻找一个平衡点.同时每个基本存储单元由几条赛道组成,以及每条赛道的长度(即可以存储的数据量)是多少,都对整个读写单元的性能有着至关重要的影响.文献[10,13]经过大量实验分析得知在多数应用中,每个基本存储单元中由 4 个条带组成,每个条带存储 64 b 数据能取得较好性能.这时如果每条赛道的读写端口大于 16,单个存储单元占用面积以及读写时延以及功耗都会急剧增加;而当读写端口数小于 16 时,单个存储单元占用面积随着端口数减少反而会增加,因为此时条带两端需要为移位操作预留的空间也越来越大.同时在后续的实验部分中,本文也分析了在本文提出的基于斯格明子的存内计算框架下读写端口的数量与移位操作数的相应变化,综合考虑我们选择读写端口为 16.

3.2 斯格明子存储单元系统优化

由于斯格明子-赛道型存储器不同于传统的 DRAM 存储器具有顺序读写的特性,因此传统的为随机读写存储器设计的系统地址映射方式并不适用于这种新型的非易失性存储器.图 9(a)所示为传统的 DRAM 的地址映射方式 RBC(row bank column),这种存储系统通常使用一种典型的开放式页面地址映射策略,将所有相邻列的同一行映射到一个连续的区域,使空间局部性最大化.同时,它通过行列交织的方式来管理流水线式的内存请求.

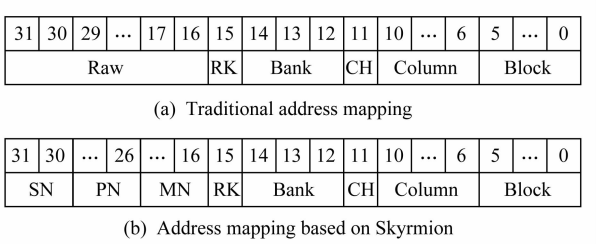


Fig. 9 Address mapping based on Skyrmion

图 9 赛道型内存地址映射方式优化

对于斯格明子-赛道型存储器来说,关键问题是传统地址映射方式将每个行作为一个连续区域而不考虑移位的问题,也不考虑这些行可能横跨了许多不同内存存储单元,因此可能会带来非常严重的负面效应.如图 10 所示,256 行依次分布在第 1 存储单元 MC1 到第 64 存储单元 MC64,为了简化讨论,假设内存中有 64 个基本存储单元,每个存储单元只有 1 条存储赛道,每条赛道只能存储 4 b 数据且只有 1 个读写端口.在传统的地址映射方式下由于内存访问都具有很高的空间局部性,导致内存访问可能在不同存储单元之间以及存储单元内部频繁切换.在图 10 的例子中,应用程序的内存访问序列为 R4→R8→R1→R4→R6.这些请求只映射到 2 个相关存储单元(MC1,MC2),从而导致了多次移位操作(总移位为 14 次).考虑到基于斯格明子的存储器存储密度极大化以及内存访问的局部性,再结合存内计算的具体应用场景,本文提出了一种新的地址映射方式,即基于斯格明子介质的地址映射方式(address mapping based on Skyrmion, AMBS).

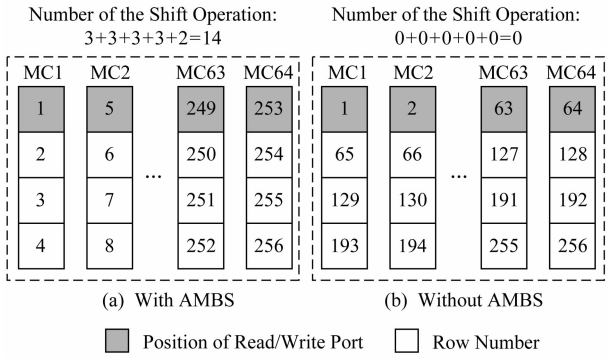


Fig. 10 An example of using AMBS

图 10 使用赛道型地址映射的优势 1 例

我们首先解释这种地址映射方案如何具体实现.AMBS 将地址位(第 16 b 到第 31 b)分为 3 部分:SN,PN,MN,见图 9(b).其中 SN(shift number)表示初始行和其对应访问端口的距离,即初始数据需要移位多少次才能够被访问.PN(port number)表示的是访问数据对应的端口序列号,即通过第几个端口去访问数据.MN(memory cell number)表示基本存储单元的编号.具体地说,如果每个 x 位数据共享一个端口,即要将内存中所有的行地址按照 SN 的数值划分为 x 组,并将地址相邻的行划为同一组.AMBS 策略优先在组内进行数据分配,只有组内整个空间分配完之后,才将后续数据分配给下一组.

通过这种赛道型内存地址映射方式可以极大地减少移位操作,原因主要有 2 方面:1)在组间来说,假设内存的总容量为 8 GB,每 32 b 数据共享一个端口,此时每组的大小为 256 MB(8 GB/32),此时由于内存读取的局部性,内存访问序列有极大可能属于某一组,因此可以减少由于较小区域的空间局部性导致的频繁移位操作.2)在组内来说,由于延迟和能耗主要来自于移位操作,特别是长距离的移位操作,因此减少移位的距离也能提高系统性能.而赛道型内存地址映射能将大部分内存读写的移位操作距离减少至 1,这是因为同一存储单元中相邻 2 行的地址差距非常大(256 MB).

结合上述说明,我们以图 10 为例来说明 AMBS 是如何工作的.内存访问序列为 R4→R8→R1→R4→R6,在传统的内存映射方式下,需要 14 次移位操作才能够读取完这些数据,而在基于赛道内存的操作下不需要进行任何移位任何操作就能完成内存数据的读取,节省了大量时间和能耗的开销.而赛道型内存地址映射方式的实现方式,可以通过在操作系统中使用一个统一的分系统管理物理页,并形成了一个层次结构基于 Shift 和 Port 的可用页面列表,类似于页面着色技术[17],根据应用程序需要的不同存储容量,尽可能地分配一个连续的区域.因此最简单的方式,可以使用一个静态的物理地址的映射系统,在内存控制器中或斯格明子-赛道型存储器芯片内部实现.这样就可以在不改变现有操作系统的存储器体系接口下实现,因此带来的额外开销也基本可以忽略.

4 实验与结果分析

本节分别从硬件层和系统层对基于斯格明子介质的存内计算单元进行性能评估.首先对于硬件层面,探讨了基于斯格明子逻辑门的存内计算单元的性能,其次在系统层面上通过通用的图像锐化程序对于内存存储单元的读写端口个数与数据移位操作数的关系,以及整个存内计算系统的时间及能耗效率进行了评估.

4.1 基于斯格明子的计算单元性能评估

斯格明子逻辑门组成的基本运算单元作为存内计算框架的基础,首先我们需要对其性能进行评估.本实验中所用的斯格明子器件的读写时间与能耗数据来自于文献[18],同时移位操作的能耗可以通过斯格明子纳米线的热耗散数据计算得出,移位操作的时间可以通过斯格明子在纳米线上的移动速度计算得出.需要注意的是当斯格明子逻辑门的工作状态即输入不同时,纳米线上的驱动电流密度也会随之变化[14-15].例如当逻辑与门的输入为 0 和 1 时(即只有一个斯格明子进入与门),电流密度为 7×10^{12} A/m^{-2};当逻辑与门的输入为 1 和 1 时(即有 2 个斯格明子同时进入与门),电路密度为 4×10^{12} Am^{-2},因此在计算整个计算单元的功耗时我们只能取其平均值.在斯格明子逻辑门中使用的纳米线长约为 600 nm,宽度约为 100 nm[14],由此我们可以计算出计算单元占用的面积.基于斯格明子计算单元对比基于磁畴壁计算单元极大地减少了额外 COMS 电路的使用,不仅使得性能上有所提高,也极大地减少了实现工艺所需的复杂度.

表 1 中对比了基于斯格明子计算单元和基于磁畴壁计算单元在时间、能耗和面积上的区别.可以看出,本文提出的基于斯格明子的存内计算单元相比目前最先进的基于磁畴壁的存内计算单元节省了 54.6%的时间、42.9%的能耗以及 23.1%的占用面积.这主要归功于斯格明子介质优异的物理性质:加法计算单元进行的优化,以及乘法计算单元对于加法器的复用.同时相比基于磁畴壁计算单元,大大减少了外围辅助电路的需求,简化了电路设计,使得基于斯格明子计算单元更容易被实现.

Table 1 Performance Comparison of Two Computing Units

表 1 两种计算单元性能比较

Functional Unit	8 b Full Adder		8 b Multiplier	
	Skyrmion	DW	Skyrmion	DW
Speed/cycle	49	108	149	326
Energy/pJ	28	40	196	308
Area/μm^{-2}	2.0	2.6	16.8	19.8

4.2 基于斯格明子的存内计算框架性能评估

4.2.1 实验环境配置

为了更准确评估基于斯格明子的存内计算框架的总体性能,本文采用调整过的基于磁畴壁的存内计算框架[1]来作为比较对象.为了模拟应用程序在存内计算框架中的具体执行过程,我们修改了体系结构模拟器 Gem5[19]中内存部分,同时为了获得具体时间和能耗数据,我们结合了功耗和时序建模工具 McPAT[20]建立了整个实验平台.

表 2 列出了实验中的主要参数配置.其中存内计算框架中主存储单元均被设置为 1 000 MHz,与计算单元保持同步.对于基于磁畴壁的存内计算框

架,时间和能耗参数可以从扩展的 NVSim[21] 中获取.由于本文提出的内存框架具有加法器和乘法器组成的计算单元,因此理论上任何程序中的加法和乘法操作均可以在此内存框架中完成.特别地,本文选取了主要操作均由加法和乘法组成的图像锐化程序作为实验测试程序,并在 4.2.2 节中介绍了图像锐化程序的具体执行过程.

Table 2 Configuration Parameters of Experiment

表 2 实验环境中关键参数配置

Configuration	Parameters
CPU	4 single x86 cores, out of order, 3 GHz
L1 Cache	32 KB I-cache, 32 KB D-cache, 4-way
L2 Cache	4 MB, 8-way, line size-64 B
Memory	Domain-wall memory@1 000 MHz, 8 GB Skyrmion memory@1 000 MHz, 8 GB

4.2.2 基于存内计算框架的图像锐化程序实例

为了详细说明程序是如何在基于斯格明子的存内计算框架中执行的,以及存内计算带来的优势,本节以图像锐化处理为例详细描述程序执行过程.

图像锐化即加强图像中重要信息,使得图像更清晰、更易于处理,在图像处理识别等各个领域都起着非常重要的作用.由于其处理的对象是以矩阵的形式将对应像素点信息存储于内存中的数字化图片,涉及到大量的矩阵操作,特别适合于用 PIM 进行并行处理.其本质是利用微分等运算加强图像中包含边缘信息的高频部分,代表性算法为拉普拉斯算子.拉普拉斯算子是一种二阶微分算子,一个连续的二元函数 $f(x,y)$ 其拉普拉斯运算定义为

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}. \tag{1}$$

对图像处理来说,可以将拉普拉斯算子简化为

$$g(i,j) = 4f(i,j) - f(i+1,j) - f(i-1,j) - f(i,j+1) - f(i,j-1). \tag{2}$$

在数字图像处理中即表示将某个点对应像素的数值乘以 4 再减去其上下左右相邻像素对应的数值.在图像锐化处理的过程中,拉普拉斯算子可以直接通过模板操作来实现,即用拉普拉斯模板与图像中对应像素数值矩阵进行点乘来得到锐化后的图像数值.如图 11 所示,其中常用的模板为

$$\boldsymbol{G}_x = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}. \tag{3}$$

图 11 以使用拉普拉斯算子模板进行图像锐化的程序为例,说明通用程序在基于斯格明子介质的存内计算框架中执行过程.如图 11 所示,基于斯格明子的存内计算框架主要包含存储单元和计算单元 2 部分.

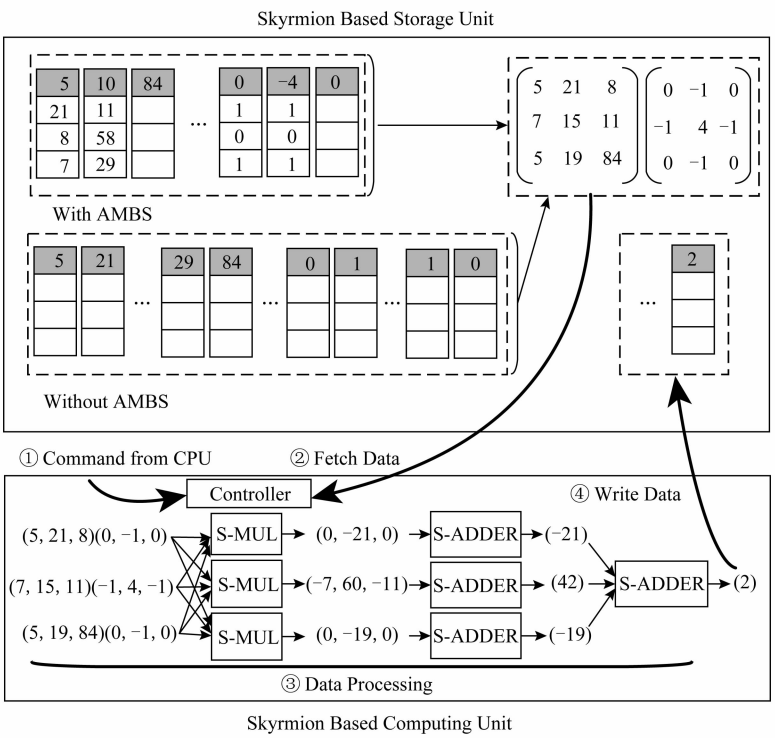


Fig. 11 The working process of image sharpening in PIM architecture

图 11 图像锐化程序在存内计算框架中具体执行过程

其中存储单元部分由斯格明子-赛道型存储器组成,如图 11 上半部分所示,灰色部分表示读写端口所在位置.在未使用 AMBS 策略之前读取 2 个矩阵的数据需要进行多次移位操作,而在使用 AMBS 策略之后数据均存储在读写端口的位置,读取这些数据不需要再进行任何移位操作.图 11 中计算单元部分由基于斯格明子的乘法器(S-MUL)和加法器(S-ADDER)等逻辑运算单元构成.同时存内计算框架不同于传统的使用外部处理器的计算框架,在内存中还应设有专门的控制器来控制程序的执行过程.如图 11 下半部分所示,图像锐化程序的执行过程可被分解为 4 个主要步骤:

步骤 1. 指令输入.此时控制指令由外部处理器输入到内部的控制器.包含需要处理的数据地址、需要进行的数据处理操作等.在如图 11 所示例子中即包含存储图片对应像素信息的 3×3 矩阵的地址、存储拉普拉斯模板矩阵的地址以及需要进行的对应矩阵元素的加分和乘法操作.

步骤 2. 取数据.内部控制器根据指令从存储单元对应地址处取出数据,在图 11 中为一个保存图像像素信息的 3×3 矩阵以及一个存储拉普拉斯算子模板信息的 3×3 矩阵.

步骤 3. 数据处理.控制器将相应数据分配至各个逻辑运算单元进行数据处理,直至得到需要的结果.图 11 中进行的是像素与模板的乘法,即 2 个矩阵的点乘运算.首先将对应矩阵按行、列进行分解,如图 11 中分解成(5,21,8)与(0,−1,0);再将对应数值输入相应的基于斯格明子的乘法器分别得到(0,−21,0);最后将结果通过基于斯格明子的加法器多次相加得到最终结果(2).

步骤 4. 数据写回.最后将处理的结果(2)再写回到存储单元中.

上述过程不断重复,直至将整个图像的数据进行类似的卷积操作之后,就可以得到锐化后的图像.在这个过程中,图像数据均不需要传输到外部处理器进行处理,从而节省了大量的时间和能耗.

4.2.3 实验结果

本实验主要分为 2 部分:第 1 部分主要分析读写端口数与移位次数之间的关系,以确定在斯格明子-赛道型存储器中基本存储单元读写端口数;第 2 部分将本文提出的基于斯格明子的存内计算框架与目前最先进的基于磁畴壁的存内计算框架进行对比.

如图 12 所示,我们首先研究了存储单元读写端口对程序执行过程中赛道型内存总移位数的影响.为了便于比较,将总移位以 Base-16 为基准进行规格化.其中 Base-X 代表不使用 AMBS 内存映射策略且每个内存单元具有 X 个读写端口的情况,AMBS-X 指的是使用 AMBS 映射策略且每个内存单元具有 X 个读写端口的情况.可以看出,使用 AMBS 内存映射策略之后移位操作次数极大减少.这主要是由于:1)AMBS 映射策略使得同一张图片的数据均被存储在同一组具有相同 SN 的内存中,因此图片锐化程序读取 1 张图片数据时就不再需要进任何移位操作;2)AMBS 映射策略使得相邻图片的数据存储在同一组或者相邻组内存中,在这种情况下读取相邻图片数据的图片也最多需要进行一次移位操作.同时我们还可以从图 12 中看出,AMBS-16 与 AMBS-32 差距较小,而与 AMBS-8 差距较大.这是由于 AMBS-16 位移操作次数已经较少,再增加端口数也无法大幅减少位移操作次数,但是减少端口数会显著增减位移操作的次数.综上所述,结合占用面积、读写延时等情况考虑,单个基本存储单元读写端口为 16 时能取得较好性能.

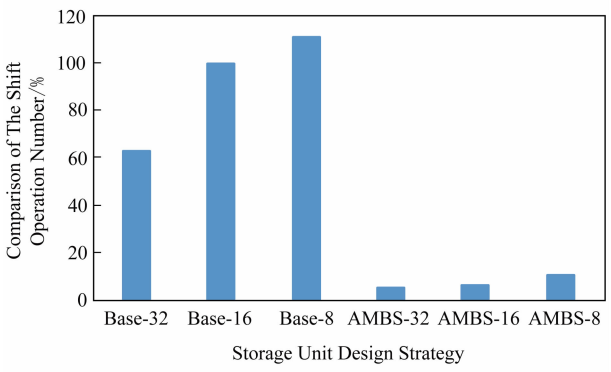


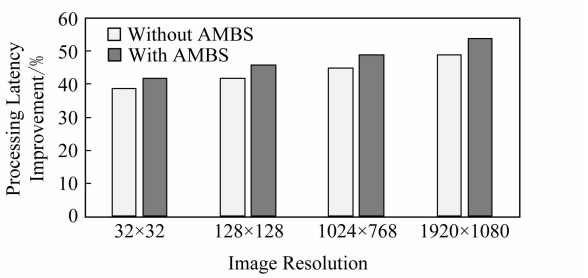
Fig. 12 The impact of the read/write ports on the shift operation compared with Base-16

图 12 与 Base-16 对比读写端口对移位操作数影响

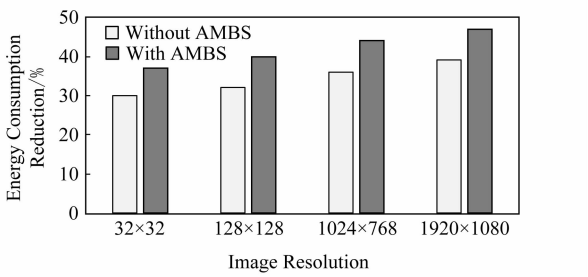
在实验的第 2 部分,我们使用通用的图像锐化程序作为实验程序,同时为了使得实验结果更具有通用性,我们使用一系列不同分辨率的图片集作为实验比较对象.

图 13 对比了在不使用 AMBS 策略(Without AMBS)与使用 AMBS 策略(With AMBS)的情况下,基于斯格明子的存内计算框架与基准性能(基于磁畴壁的存内计算框架)的比较.可以计算得出,使用 AMBS 策略与不使用 AMBS 策略相比,存内计算框架平均能节省 4.5%的时间和 8.7%的能耗.

容易观察到,系统整体性能的差距要比图 12 中位移次数的差距小得多.这主要是因为在基于斯格明子的存内计算框架中计算单元占据了大部分的时间和能耗.同时我们也注意到当测试图片逐步增大时,时间和能耗的减少比例也逐步扩大并趋近一个极限值.这是由于随着数据量的不断增加,斯格明子内存框架内程序运行的并行程度也在不断提高并接近其极限.因此在一定范围内,斯格明子存内计算框架中程序处理的数据量越大越能获得更多优势.



(a) Comparison of time consumption at different image resolutions



(b) Comparison of energy consumption at different image resolutions

Fig. 13 Performance evaluation of PIM architecture based on Skyrmion

图 13 基于斯格明子的存内计算框架性能评估

从总体上来说,实验结果表明:在不使用 AMBS 映射策略下,本文提出的基于斯格明子的存内计算框架相比目前最先进的基于磁畴壁的存内计算单元在时间上平均节省了 43.6%,在能耗上平均节省了 34.2%.在使用了 AMBS 映射策略之后,平均节约时间上升至 48.1%,同时平均节约能耗 42.9%.

5 总 结

本文提出了基于斯格明子逻辑门的加法和乘法计算单元,探讨了斯格明子基本存储单元的设计方式,优化了斯格明子存储单元的地址映射方式,并最终在此基础上建立了基于斯格明子介质的存内计算框架.本文提出的存内计算框架在获得基于斯格明子-赛道型内存的非易失性存储单元优势的同时又获得了基于斯格明子逻辑计算单元的优势.在存储单元方面,本文首先从硬件层面探讨了斯格明子-赛道型存储单元的读写参数优化等问题,再从系统层面提出了基于斯格明子-赛道型存储单元专用内存映射策略,从而在总体上改善了存内计算单元的性能.在计算单元方面,本文提出的基于斯格明子的全加器和乘法器不仅受益于斯格明子本身优异的物理特性,同时计算单元的并行优化设计以及电路的复用也极大地提高了系统整体性能,降低了系统实现的复杂度.实验表明:本文提出的存内计算框架与目前最先进的基于磁畴壁的存内计算框架相比,在时间上平均节省了 48.1%,在能耗上平均节省了 42.9%.

参 考 文 献

[1] Luo Le, Liu Yi, Qian Depei. Survey on in-memory computing technology [J]. Journal of Software, 2016, 27(8): 2147-2167 (in Chinese)
(罗乐, 刘轶, 钱德沛. 存内计算技术研究综述[J]. 软件学报, 2016, 27(8): 2147-2167)

[2] Gokhale M, Holmes B, Iobst K. Processing in memory: The Terasys massively parallel PIM array [J]. Computer, 1995, 28(4): 23-31

[3] Zhang Dongping, Jayasena N, Lyashevsky A, et al. TOP-PIM: Throughput-oriented programmable processing in memory [C] //Proc of the 23rd ACM Int Symp on High-Performance Parallel and Distributed Computing. New York: ACM, 2014: 85-98

[4] Akin B, Franchetti F, Hoe J C. Data reorganization in memory using 3D-stacked DRAM [J]. ACM SIGARCH Computer Architecture News, 2015, 43(3): 131-143

[5] Joshi M, Zhang Wangyuan, Li Tao. Mercury: A fast and energy-efficient multi-level cell based phase change memory system [C] //Proc of the 17th High Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2011: 345-356

[6] Kültürsay E, Kandemir M, Sivasubramaniam A, et al. Evaluating STT-RAM as an energy-efficient main memory alternative [C] //Proc of the Performance Analysis of Systems and Software (ISPASS). Piscataway, NJ: IEEE, 2013: 256-267

[7] Lee B C, Ipek E, Mutlu O, et al. Architecting phase change memory as a scalable DRAM alternative [J]. ACM SIGARCH Computer Architecture News, 2009, 37(3): 2-13

[8] Parkin S, Yang Seehun. Memory on the racetrack [J]. Nature Nanotechnology, 2015, 10(3): 195-198

[9] Mao Mengjie, Wen Wujie, Zhang Yaojun, et al. Exploration of GPGPU register file architecture using domain-wall-shift-write based racetrack memory [C] //Proc of the 51st Annual Design Automation Conf. New York: ACM, 2014: 1-6

[10] Hu Qingda, Sun Guangyu, Shu Jiwu, et al. Exploring main memory design based on racetrack memory technology [C] // Proc of the 26th Great Lakes Symp on VLSI. Piscataway, NJ: IEEE, 2016: 397-402

[11] Yu Hao, Wang Yuhao, Chen Shuai, et al. Energy efficient in-memory machine learning for data intensive image-processing by non-volatile domain-wall memory [C] //Proc of the 19th Asia and South Pacific Design Automation Conf (ASP-DAC). Piscataway, NJ: IEEE, 2014: 191-196

[12] Fert A, Cros V, Sampaio J. Skyrmions on the track [J]. Nature Nanotechnology, 2013, 8(3): 152-156

[13] Tomasello R, Martinez E, Zivieri R, et al. A strategy for the design of Skyrmion racetrack memories [J/OL]. Scientific Reports, 2014: Article number 6784. [2018-08-02]. https://www.nature.com/articles/srep06784

[14] Zhang Xichao, Ezawa M, Zhou Yan. Magnetic Skyrmion logic gates: Conversion, duplication and merging of Skyrmions [J/OL]. Scientific Reports, 2015: Article number 9400. [2018-08-02]. https://www.nature.com/articles/srep09400

[15] Xing Xiangjun, Pong Philip, Zhou Yan. Skyrmion domain wall collision and domain wall-gated Skyrmion logic [J]. Physical Review B, 2016, 94(5): 054408

[16] Sun Guangyu, Zhang Chao, Li Hehe, et al. From device to system: Cross-layer design exploration of racetrack memory [C] //Proc of the Design, Automation and Test in Europe. San Jose, CA: EDA Consortium, 2015: 1018-1023

[17] Jeong M K, Yoon D H, Sunwoo D, et al. Balancing DRAM locality and parallelism in shared memory CMP systems [C] //Proc of the High Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2012: 1-12

[18] Kang Wang, Huang Yangqi, Zheng Chentian, et al. Voltage controlled magnetic Skyrmion motion for racetrack memory [J/OL]. Scientific Reports, 2016: Article number 23164. [2018-08-02]. https://www.nature.com/articles/srep23164

[19] Binkert N, Beckmann B, Black G, et al. The Gem5 simulator [J]. ACM SIGARCH Computer Architecture News, 2011, 39(2): 1-7

[20] Li Sheng, Ahn J H, Strong R D, et al. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures [C] //Proc of the Annual IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2009: 469-480

[21] Dong Xiangyu, Xu Cong, Jouppi N, et al. NVSim: A circuit-level performance, energy, and area model for emerging non-volatile memory [J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2012, 31(7): 994-1007

Liu Bicheng, born in 1988. MEn. His main research interests include computer architecture, processing in memory, and computer architecture.

Gu Haifeng, born in 1991. PhD candidate. His main research interests include hardware/software co-validation, design automation of embedded systems, and software engineering.

Chen Mingsong, born in 1982. PhD, professor, PhD supervisor. Senior member of CCF. His main research interests include design automation and verification of cyber physical systems, cloud/embedded/parallel computing, computer architecture, and IoT Security.

Gu Shouzhen, born in 1987. PhD, lecturer. Member of CCF. Her main research interests include non-volatile memory, storage architecture, and embedded/parallel computing.

Chen Wenjie, born in 1977. PhD, associate professor. Senior member of CCF. His main research interests include software design for embedded real-time systems, logic design, and computer architecture.