中国科学院大学
University of Chinese Academy of Sciences

高级操作系统教程

文献阅读报告

题目：存算一体机器人加速器

姓名：孙文昊

学号：2023E8013282127

单位：计算技术研究所

2023 年 11 月

目　　录

高级操作系统课程文献阅读报告

报告人介绍

孙文昊，中科院计算所智能计算机研究中心硕士生，目前的研究兴趣和内容主要包括存算一体架构、工业智能计算机、处理器并行等。

调研方向介绍

主要调研存算一体技术的提出、发展过程、基本路线、基本原理，并以 Dadu 系列处理器为例，剖析存算一体处理器的工作原理。

近年研究介绍

存算一体技术是近年来备受关注的研究方向，在计算范式上，主要从近存计算、存内计算两个方向开展研究；在材料器件上，主要围绕忆阻器（RRAM）开展相关研究。

国内外学术界和产业界都在进行相关研究，如三星电子、SK 海力士、台积电、美光、IBM、英特尔等。国内的研究机构包括中科院计算所、中科院微电子所、清华大学电子系与集成电路系、复旦大学芯片院等。

这些机构的研究人员在存算一体技术的发展和应用方面做出了重要贡献，如刘明院士、汪玉教授、陈晓明副研究员、陈迟晓副研究员等。存算一体技术的主要成果包括：减少不必要的数据搬运，使用存储单元参与逻辑计算提升算力，降低成本等。存算一体技术在人工智能、元宇宙等领域有着广泛的应用前景。

存算一体技术主要在计算机体系结构、集成电路设计相关的会议和期刊发表文章：

会议：ISSCC、ISCA、MICRO、ASPLOS、HPCA、DAC、ICCAD 等

期刊：Science、TCAD、TC、TVLSI 等

存算一体机器人加速器

摘 要：随着摩尔定律的快速发展，性能剪刀差与存储墙问题日益严重。存算一体作为有望突破“存储墙“限制的新兴计算技术，能够提供更高的矩阵运算效率、更快的访存速度。本文首先介绍了存算一体的提出背景与基本路线、原理，然后以 Dadu 系列处理器为例，以 Dadu-cd 处理器为重点介绍了存算一体技术在机器人算法加速中的应用，最后对存算一体技术进行总结与展望。

关键词：存算一体、机器人加速器、碰撞检测

1 研究背景与动机

1.1 冯诺伊曼架构“存储墙”问题

计算机技术的不断进步促进了现代信息技术的飞速发展，使人类社会的生产生活方式发生了巨大的变化。既有计算设备的体系结构一直沿用的是上世纪 40 年代由 John von Neumann 提出的冯诺依曼架构[1]，其核心思想为“存算分离”和“存储程序”。

“存储程序”即数据和指令均被视为数据，存放在存储单元中；“存算分离”即将计算资源和存储资源进行空间划分。其中，计算单元实现数据计算，存储单元负责储存指令与数据。只有当计算单元需要相应的数据或指令时，才会由控制单元将其从存储单元中搬运到计算单元中。等待完成计算后，再写回存储单元。冯诺依曼体系结构的结构逻辑如图 1.1 所示。

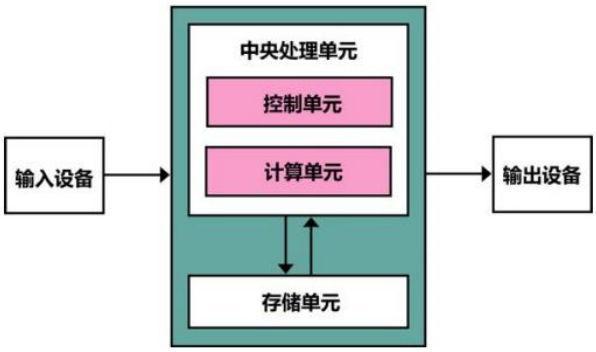

图 1.1 冯诺伊曼架构

冯诺依曼体系结构使得计算机的设计和控制高度结构化和自动化，降低了计算机设计的复杂度，然而，与之对应会同时带来计算单元与存储器件间频繁的数据搬运，进而产生计算效率下降、带宽受限等瓶颈，引发“存储墙(Memory Wall)”问题[1]。引发存储墙问题的根本原因是处理器和内存的速度差异。根据摩尔定律(More ‘s Law)的揭示，处理器每 18-24 个月的速度会提升一倍，即每年增长约 60%的性能。然而，以 DRAM(Dynamic Random Access Memory)为代表的主存性能每年只增长约 7%，导致每年产生约 50%的性能“剪刀差”，如图 1. 2 所示。高速处理器难以获得可与之性能匹配的访存数据流和指令流，使得数据在存储单元和计算单元之间的搬运成为了限制冯诺依曼架构的主要瓶颈之一，因此，“存储墙”问题也称为“冯诺依曼瓶颈”。

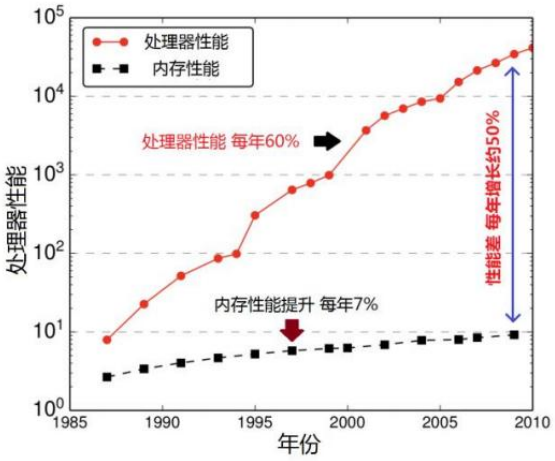

图 1.2 “存储墙”导致的性能“剪刀差”

近年来，为了缓解“冯诺依曼瓶颈”，体系结构研究者在冯诺伊曼架构上做出了多种改进与尝试，主要的解决思路有两种。第一种是充分利用存储单元“大和快”的矛盾，利用数据访问的的局部性，使用不断加深的存储层次来“隐藏”访存延迟，达到提升系统性能的目的。这种方法被称为“存储分层(memory hierarchy)”。

例如，片上高速缓存(Cache)的引入被认为是使用存储分层缓解“存储墙”问题的一个有效解决手段。Cache 是内存数据映射的一个子集，它根据数据的时间和空间局部性，将近期访问的和可能被访问的数据缓存在 Cache 中。由于 Cache 的访问速度介于内存和寄存器之间，通过先查找 Cache、后访问内存的步骤，可

以有效的减少访问内存的次数，降低访存延迟。为了平衡 Cache 的大小、访问延迟和数据共享程度等特性，体系结构研究者又提出了多级 Cache 架构，进一步加深了存储的层次深度，并最终演变为经典的“处理器-三级缓存-内存-硬盘”架构。例如，在 Intel Core i9-9980XE 处理器中就采用了三级片上缓存的设计方式，且最后一级缓存(last-level cache, LLC)的大小已经达到 24.75MB。如此巨大的 LLC 使得片上处理器可以在更广的内存空间中挖掘数据的局部性。第二种方式是数据预取(prefeching)，其技术路线和第一种方式相同，也是“隐藏”访存延迟。与之不同的是，预取是通过记录一段时间内的访存历史信息或执行情况，在访存发生前将可能访问的数据提前加载到片上，从而达到隐藏访存延迟的目的。“存储分层”与预取两种方式相辅相成，共同成为了片上多核/众核处理器体系结构中“隐藏”访存延迟的重要技术。

综上，基于“存算分离”和“存储分层”理念设计的传统计算机体系结构无法从本质上解决“存储墙”问题。近年来，随着大模型、大数据、云计算、深度学习、等信息技术的迅猛发展，信息处理已由“计算密集型”向“存储密集型”转移。“存储密集型”应用具有以下两个特点：

一、数据集容量大。数据表明，当今社会每天产生约 12.5 万亿字节的数据量，且这个数据还在不断增长。海量的数据体现在对体系结构层面的影响上，则表现为对访存带宽的高需求。

二、访存离散化和随机化。由于数据集大小的扩大，数据的内存占用也随之扩大，导致数据访问的局部性变差。在这种情况下，片上缓存不断发生缺失(miss)，访存次数不断增多，使得计算机系统的“存储墙”问题愈发凸显，难以满足对海量数据的处理需求。举例说明，有数据表明，传统计算机系统在执行金融欺诈检测(financial fraud detection, FD)类程序时，其 LLC 的命中率不足 3% 。

因此，计算机科学研究者们认为，要从根本上解决“存储墙”问题，需要在底层的架构上获得新的突破。

1.2 机器人控制规划处理器

机器人相关算法中有许多不同的计算任务，如运动规划和机器人控制算法。随着机器人技术的不断发展，机器人的结构越来越复杂，机器人也将面临更加复

杂的环境和任务。结构复杂，增加了规划和控制的计算量。在复杂的环境中，不可预测的障碍和动态变化的环境将是运动规划算法需要处理的挑战。对于移动机器人而言，其电池寿命是一个非常重要的性能指标，如何在保证机器人系统实时性的同时快速获得规划结果，使算法的功耗和能耗最小化，也随之为一个值得研究的重要问题[2]。

碰撞检测的性能通常是整个机器人运动规划的瓶颈。实际的运动规划通常涉及大量的碰撞检测。目前，常见的机器人控制规划处理器包括传统 CPU、GPU 以及 FPGA 三种，各有利弊。

对于 CPU 来说，通用处理器并不能高效完成数据量较大的任务，以碰撞检测任务为例，快速探索随机树（RRT）在 CPU 上需要执行大量的时间，同时 99% 的指令会被用于碰撞检测，极大地降低了其他任务的处理性能。

对于 GPU 来说，由于碰撞检测需要同时参与运算的数据量很大，GPU 运行 RRT*算法的时间也需要 7.719 秒，无法满足机器人系统对实时性的要求；与此同时，作为边缘端设备，高能耗的 GPU 对供电的要求很高，这无疑对机器人系统的设计与实现增加了更多的不确定性。

因此，像 CPU 和 GPU 这样的通用处理器通常在处理运动。然而，上述设备在规划算法时效率很低，特别是在实时应用程序中。以快速探索随机树(RRT)基本运动规划算法为例。当它在 CPU 上实现时，99%的指令被执行用于碰撞检测，占总计划时间(约200秒)的90%以上。即使使用高性能的NVIDIA GTX Titan GPU，运行 RRT*算法[4](RRT 的改进版本)也需要 7.719 秒，其中有 7169 个碰撞检查。对于另一种流行的运动规划算法，概率路线图(PRM)中，GPU 可以获得比 CPU 实现高 10-100 倍的性能，运行时间通常在 1s。尽管如此，这样的性能仍然不能满足具有运动规划的实时应用程序。当移动机器人的电力供应有限时，问题更为严重。

虽然目前已经广泛使用 GPU 来加速碰撞检测，但 GPU 的能耗很高。与此同时，FPGA 也被考虑过，科研工作者在 FPGA 上实现了宽相位碰撞检测，其性能优于当前 CPU 上的最先进库。基于 FPGA 的 RRT*加速器比 CPU 实现的加速快 30 倍以上。在 FPGA 上实现了基于体素的碰撞检测。它将路线图中的每条边编

码为门级逻辑实现，可以同时检查以实现惊人的加速但基于 FPGA 的碰撞检测加速器有一个主要的限制：即 FPGA 的片上 RAM 太小，无法存储大型地图。

2 存算一体技术

存算一体，顾名思义，是将计算嵌入在存储系统中，避免数据在存储、计算两个模块之间频繁的传输。体系结构研究者在上世纪 90 年代时提出了一种新的解决思路：在内存中集成计算核心并承担计算任务，实现数据的“近数据”计算。这一思路被称存算一体思想，主要包括存内计算体系结构(Computing-in-Memory, CIM)，以及近存计算(Computing-near-Memory, CNM)两种状态。其中，近存计算和存内计算分别有以下特点：

- 近存计算：将计算单元置于离存储器更近的位置，本质上仍然存算分离，减少数据传输距离、提高访存带宽、降低访存延迟；
- 存内计算：在存储器中原位执行计算，减少（甚至消除）数据传输、大幅提升计算并行度。

图 2.1 对比了 CIM 架构与传统冯诺伊曼架构，不难看出：CIM 处理器实现了计算、访存的一体化，减少了数据迁移的环节，有独立的权重缓冲和计算宏[3]。

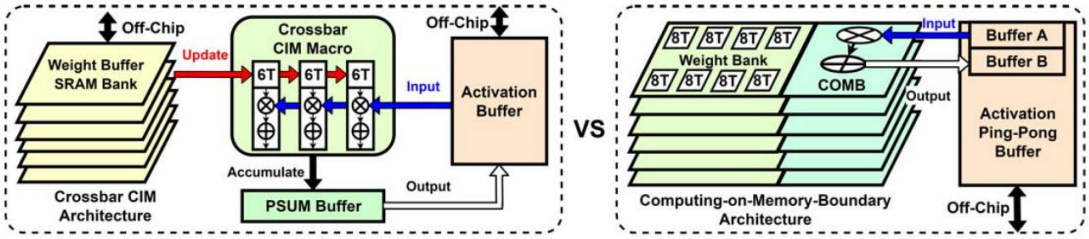

图 2.1 存算一体的发展过程

最近，已经提出了几种创新的 CNM 和 CIM 系统，包括领域特定架构，如 Neurocube、ISAAC、Microsoft Brainwave NPU 以及一些 DNN 加速器等。这些系统比通用冯·诺伊曼机器快几个数量级，能效比更高，但仅针对特定的应用领域。UPMEM 已经展示了在更通用的现成系统中进行内存中处理（PIM）的案例研究。最近，三星和海力士提出了基于 HBM2 和 GDDR6 DRAM 标准的面向机器学习的 CNM 系统，支持 TFLOPS。在 CIM 方面，仅在过去几年里，

主要的内存供应商，如三星、台积电和 IBM，已经制造了基于记忆电阻和磁性技术的 CIM 芯片，获得了无与伦比的性能和能源效率。这是计算机架构领域的重要里程碑，因为就在几年前，这些系统被认为过于奇特和难以制造。

2.1 近存计算

近存计算是一种新兴的计算模式，它将计算能力直接嵌入到数据存储设备附近，以减少数据传输延迟和提高计算效率。这种计算模式旨在解决传统计算机体系结构中存在的数据传输瓶颈问题，其中计算操作通常需要从存储设备中获取数据，导致了延迟和能源浪费。近存计算的实施方式包括使用硬件加速器、特殊的处理单元和存储设备内部的计算单元。同时，近存计算还需要新的编程模型和算法优化，以充分发挥其性能优势。

总的来说，目前已有四几种比较成熟的近存计算存算体系结构，如图 2.2，其中包括：

- 近阵列计算：在存储阵列外围电路中添加计算逻辑，只能实现特定逻辑，高并行，但要求数据对齐；
- DIMM 近存计算：在 DRAM 芯片中集成处理器，可实现通用计算，但性能较弱；
- 三维堆叠近存计算：利用三维堆叠技术，将多个 DRAM 芯片和一个逻辑芯片垂直堆叠在一起，可实现通用计算，且性能比 DIMM 近存强；
- SSD 近存计算：在 SSD 主控中集成计算资源，原理上与 DIMM 近存类似。

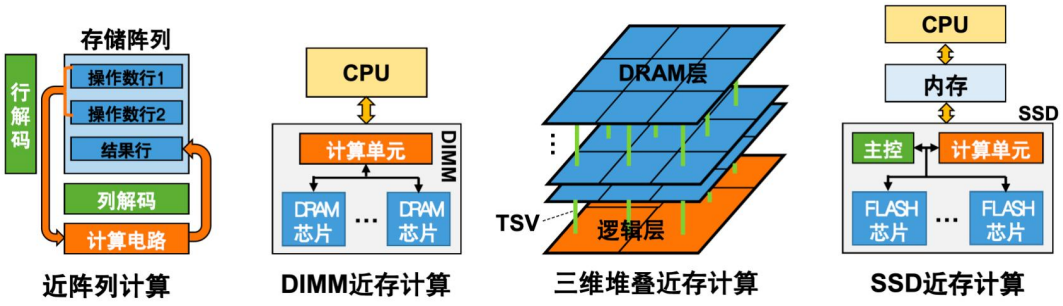

图 2.2 主流近存计算结构

这些新架构的广泛采用将在很大程度上受到软件生态系统的影响。在 CIM 和 CNM 系统中，将计算映射到计算设备对性能和能源效率至关重要。然而，包

括商用的 UPMEM 架构在内，大多数系统今天提供低级设备库，并将映射问题和优化留给开发者。这使得这些设备的可编程性和可操作性成为严重挑战。

2.2 存内计算

存内计算也是一种基于存算一体思想的计算机架构和计算模式，与近存计算不同，存内计算的计算操作直接在内存中执行，而不是从内存中获取数据传输到中央处理单元（CPU）进行计算，以此真正实现了“存”和“算”的融合一体化。这种计算模式的目标是减少数据传输延迟，提高计算效率，特别适用于大规模数据处理、实时分析和高性能计算领域。存内计算技术的实现可以涉及硬件加速器、专门设计的处理单元和编程模型的创新。一些存内计算系统还利用了内存层次结构，以实现高性能和低延迟的计算。

存算阵列（例如 RRAM）在执行模拟信号矩阵向量乘法时，可以在一个非常短的时间内（常数时间）完成计算，无论输入矩阵和向量的大小有多大。

"存算融合" 意味着将存储和计算功能融合在一起，其中存储器件不仅用于存储数据，还可以直接参与乘法计算操作。这种融合改变了传统计算机体系结构中存储和计算分离的模式，提供了更高效的计算方式。

通常，传统计算机体系结构中的计算操作需要从存储设备中获取数据，然后传输到中央处理单元（CPU）进行计算。这个过程需要大量的数据传输，会引入延迟和能源消耗。但在存算融合中，存储器件本身可以执行一部分计算，而不需要将数据复制到 CPU。

在具体的应用中，存算融合可以采用不同的技术，其中一种可能的技术是使用非易失性内存（NVM）或电阻式随机存取存储器（RRAM）等存储技术，它们允许在存储设备内部进行计算操作。这种方式有助于减少数据传输，降低计算延迟，提高能源效率，并在一些情况下提供更快的计算速度。

"存算融合" 在数据密集型应用中特别有潜力，如人工智能、大数据分析、深度学习等领域，因为这些应用通常需要大量的数据处理和计算。通过将存储和计算融合在一起，可以提高计算效率，加速处理速度，并节省能源。

总之，"存算融合"是一种创新的计算方式，允许存储器件不仅用于数据存储，还可以参与计算操作，从而提高计算效率和性能。这一领域仍在不断发展，可能对未来的计算机技术产生深远影响。

可以通过与基尔霍夫定律类比理解存内计算的思想。在一个电路中，流入某一节点的电流等于流出该节点的电流的总和。这可以类比为存内计算中的数据流动。在存内计算中，数据可以看作是电流，而存储设备和计算单元可以看作是电路中的节点。基于电流定律，可以将数据流在存储和计算单元之间进行，从而减少数据传输，提高效率，就像电流在电路中保持平衡一样。

2.3 忆阻器

RRAM（Resistive Random-Access Memory）通常被称为"忆阻存储器"，也有人称之为"电阻式随机存取存储器"，简称为“忆阻器”。它是一种基于电阻变化来存储数据的非挥发性存储器技术。 RRAM 的工作原理涉及通过改变电阻状态来存储和读取数据，这些状态可以代表二进制位（0 和 1）。因此，RRAM 技术可以被归类为忆阻器技术，而忆阻器是一种根据电阻状态来存储信息的设备。RRAM 技术在存储和内存领域具有广泛的应用潜力，因为它具有高速、低能耗、非挥发性和长寿命等优点。

RRAM 的工作原理基于一种叫做"电阻突变"的现象。每个 RRAM 存储单元包含一个非晶硅或氧化物等电阻层，夹在两个电极之间。当施加适当的电压时，电阻层中的电阻会发生可逆变化，可以切换存储单元的状态，表示为 "0" 或 "1"。忆阻器的工作原理如图 2.3。

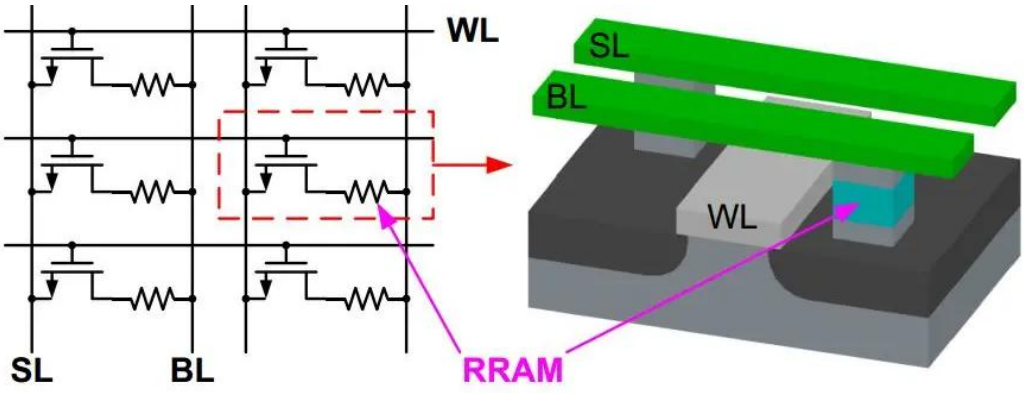

图 2.3 RRAM 工作原理

通过改变电压，可以在 RRAM 存储单元中存储不同的电阻状态，这对应于存储的二进制数据。读取操作通过测量存储单元的电阻状态来实现。电阻状态的变化是由于在电阻层中的离子迁移或电子位置改变引起的。RRAM 技术有望在未来的存储器和内存技术中发挥关键作用，因为它具有许多优势，包括快速、低能耗、非挥发性和长寿命。这使得它适用于各种应用，从嵌入式系统到大规模数据中心。

2023 年, 清华大学集成电路学院提出了一款名为 STELLAR 架构的忆阻器[4]。STELLAR 架构是一种基于符号和阈值的学习架构，可以在芯片上进行完全的在线学习，无需外部存储器的辅助。该架构避免了复杂的权重更新计算和写入验证过程，降低了电路设计难度和开销，提高了学习效率和准确度。STELLAR 架构采用循环并行导电率调节方案实现行级并行的导电率调节, 进一步降低了能耗和延迟，并缓解了忆阻器寿命的限制。

STELLAR 架构的优势在于它能够在芯片上实现在线学习，避免了复杂的权重更新计算和写入验证过程，降低了电路设计难度和开销，提高了学习效率和准确度[2]. 与传统的反向传播算法相比，该架构在模拟 MNIST 数据集分类任务时，能够节省两个数量级的能耗。该架构采用循环并行导电率调节方案实现行级并行的导电率调节，进一步降低了能耗和延迟，并缓解了忆阻器寿命的限制。

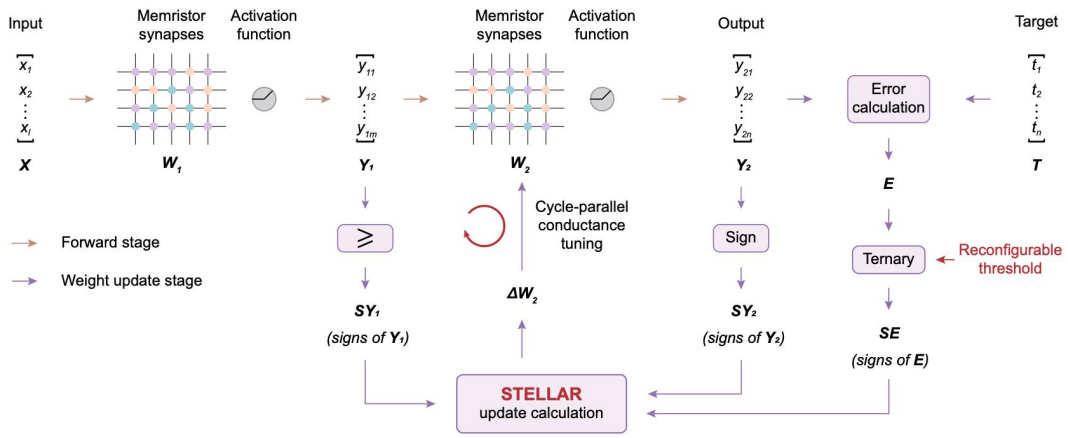

图 2-4 STELLAR 架构

3 机器人碰撞问题建模

3.1 三维空间的建模

三维模型可以用不同的体积方法来表示。最简单的表示是体素图。一个三维空间被分割成等大小的体素，整个空间可以用一个三维矩阵表示，其中每个矩阵元素代表一个体素。最简单的编码方案是用一位来表示体素是被占用的还是空的。这种简单的表示方式会占用大量内存。为了节省内存消耗，可以将体素映射转换为体素列表，这是映射中已占用的体素的下标列表。体素列表是表示稀疏地图的有效方法。

一般来说，通过八叉树的数据结构实现数据表示的思路更为常见。在这个方法中，一个立方体空间被划分为 8 个 18 个子立方体。这些子立方体递归地划分，直到它们达到最佳分辨率，或者它们完全空或被占用，如图 3.1 所示。

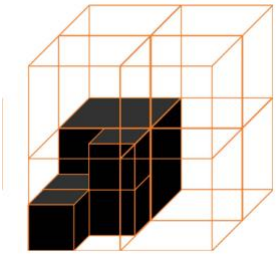

图 3.1 三维空间的立方体表示

八叉树也是一种层次树，其中一个多维数据集对应于一个节点，多维数据集的每个子多维数据集都作为该节点的子节点插入，如图 3.2 所示。

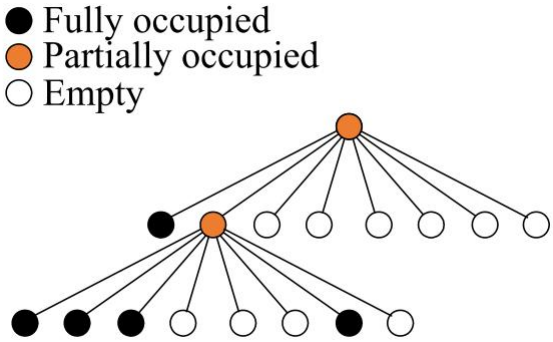

图 3.2 立方体的八叉树表示

八叉树中的每个节点表示相应的子立方体空间的状态(包括完全占用、部分占用和空的)。只有部分占据的节点有子节点，对应于部分占据的子立方体需要

进一步分区，直到最细的体素被完全占据或为空。八叉树比体素列表 (Voxel Lists) 更适合动态环境，因为后者是固定的，无法根据环境变化进行灵活的修。OctoMap[6]是一个广泛使用的基于八叉树表示的三维建模库。它使用 2 比特对节点进行编码:10 为空节点，01 为已满节点，00 为未知节点，11 为部分已满节点。在这项工作中，作者采用基于此建模库对空间进行八叉树表示。

3.2 空间碰撞的建模

碰撞检测的算法建立在碰撞模型的表示上。它递归地检查两棵八叉树中的两个节点，直到找到一个碰撞或达到无碰撞。两个节点之间的碰撞检测是通过检查它们的子节点，即两个 16 位向量来实现的。对于特定的节点编码方案，两个节点之间的碰撞检测可以通过在两个 16 位向量之间进行一些布尔逻辑来实现。如果不能确定两个节点是否冲突，这意味着两个节点都部分被占用，那么就需要递归地检查它们的子节点。

两棵八叉树之间的碰撞检测具有有限的并行性，因为它是一个递归流，在算法的任何阶段，只能检查两个节点的子节点的碰撞。为了解决这个问题，一种可能的方法是让节点有更多的子节点，从而带来更高的并行度。例如，每个节点有 64 个子节点，并行度增加 8×。这还意味着一个立方体空间被划分为 4×4×4=64 个子立方体，这也相当于对八叉树进行修改，将八叉树中的 2 个层次扁平化为 1 个层次。这种树型结构称为 VOLA。

然而, VOLA 的存储效率较差。当分辨率较高且最优体素的大小相对较小时，VOLA 将比八叉树消耗更多的内存。例如，在一个只有一个体素的 4×4×4 立方体空间中，VOLA 需要 65 个节点来编码空间，而八叉树只需要 17 个节点。在最坏的情况下，VOLA 的存储需求(对于 4×4×4 分区方案)几乎达到了八叉树所需内存的 4 倍。为了这种方法适用于内存处理，需要通过重新设计处理芯片解决并行性和存储效率问题。

3.2 DRAM 的结构

一个主存储器系统会包括若干个 DRAM 芯片。图 3.3 显示了 DRAM 芯片的高级架构。一个动态随机存取存储器芯片由多个银行和 I/O 电路组成。芯片中的

所有银行共享一个内部总线来读写数据。每个银行依次包含多个子数组和一个行缓冲区。每个子阵列由数百行共享一排感测放大器(SAs)的 DRAM 单元组成。sa 还充当每个子数组中的本地行缓冲区。每个子数组的典型内存数组大小为(512 或 1024)×(8192 或 16384)。但是，外部 DRAM 总线的宽度非常窄，通常是 64 位。在许多内存密集型应用程序中，内部操作宽度和外部总线宽度之间的巨大差异导致了内存壁瓶颈，包括本工作中考虑的运动规划。

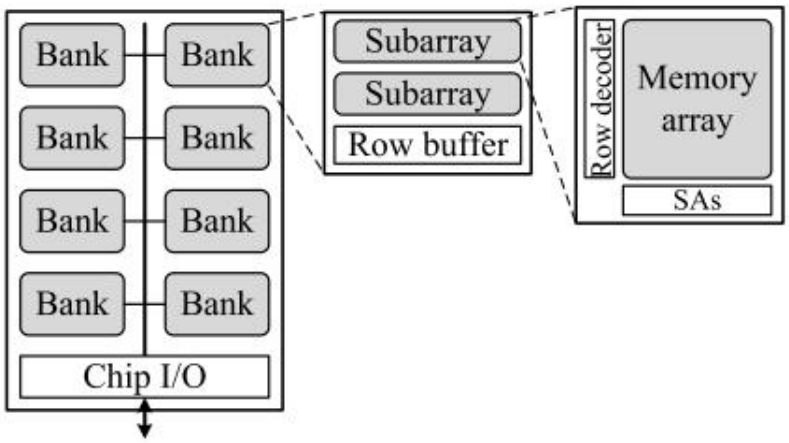

图 3.3 DRAM 芯片架构

为解决 3.2 节这提到的 VOLA 编码内存不足的问题，可以将一些处理逻辑集成到 DRAM 中，命名为 PIM。因此，支持的操作在 DRAM 内部进行，而不需要将数据传输到 CPU，从而可以提高性能和能源效率。

4 用于碰撞检测的存算一体架构

4.1 问题需求

根据前文分析, 碰撞检测是一个内存密集型问题, 传统的基于 ASIC 和 FPGA 加速器[5]会收到内存带宽限制。文章提出了一个 PIM 体系结构来加速在 3.2 中所述的碰撞检测问题。

设计的基本动机是充分利用 DRAM 的内部带宽，消除内存带宽瓶颈，从而为碰撞检测提供更高的性能和能效。这里的创新包括硬件和软件技术。在硬件方面，需要修改 DRAM 架构，以支持碰撞检测的 PIM 操作。

相应的修改在图 4.1 中以蓝色模块表示。其中：在每个 DRAM 模块中添加一个 PIM 逻辑模块，在每个模块内部进行所涉及的操作。在每个子数组中添加了 3 个额外的内存行以及一个额外的单词行(WL)驱动程序。额外的内存行用于执行内存中的 AND 操作。

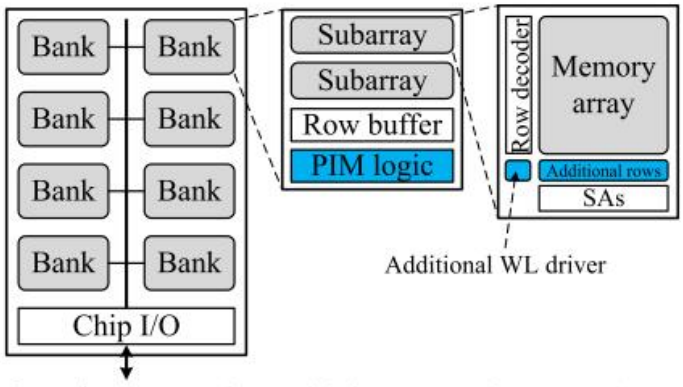

图 4.1 修改后的 DRAM 架构

可以看到，PIM 体系结构利用了 bank 宽度级别的高带宽，并有望提供大规模的并行性。注意，经过的修改后，DRAM 仍然可以进行正常的数据读写。在软件方面，为了支持硬件提供的大规模并行性，提出了一种灵活的八叉树数据结构，它不仅适合大规模并行的 PIM 体系结构，而且减少了递归，提高了存储效率。此外，在灵活的八叉树数据结构中提出了一种新的节点编码方案。详细阐述了灵活八叉树中节点状态的编码, 通过简单的批量位与运算完成节点间的碰撞检测，并在每个子阵中实现。

与之对应的是，在 Dadu-P[5]中，采用的是传统的 ASIC 思路：即用 FPGA 实现机器人的碰撞检测加速，如图 4.2。这种策略会面临前文所述的“存储墙”问题，尤其是在面对 3.2 中的八叉树编码时，难以高效及时处理海量数据。

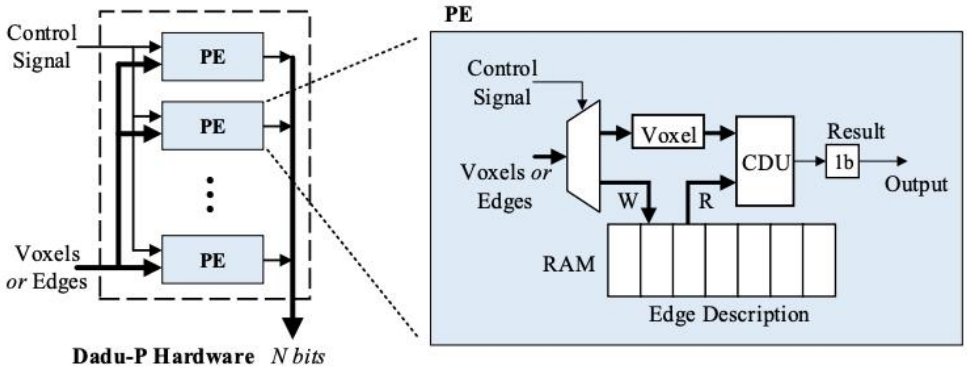

图 4.2 Dadu-P 的架构设计

4.2 八叉树数据结构与节点编码

为支持 DRAM 内部的大规模并行处理，对八叉树数据结构的一个简单修改就是让节点有更多的子节点。但这种方法的缺点是存储效率低。在大多数情况下，两棵八叉树之间的碰撞检测是在八叉树的前几级完成的。这可以直观地解释如下：

对于机器人来说，路线图中的一条边(对应于机器人的运动路径)就是机器人的扫掠体积。它是一个实心形状，因此大多数相应的八叉树节点都被占用了。用其他节点检查已占用的节点很容易，因为只有两种可能的结果:碰撞或无碰撞，而且不需要递归检查。

受上述观察的启发，通过在第一级(对应于根节点的子节点)选择适当的分区粒度，在大多数情况下，碰撞检测在第一级完成。提出了一种灵活的八叉树，它具有较高的并行性和可接受的存储效率。其主要思想是使根节点的子节点数量灵活，同时保持其他节点不变。灵活八叉树不仅减少了递归，而且增加了并行性。高并行性是通过并发处理根节点的子节点来实现的，这比原来的八叉树消耗了更多的内存。

更具体地说，需要首先找到一个大小合适的空间 W0，确保 O⊆W0⊂W，由灵活八叉树的根节点表示。然后将根空间划分为大量的子立方体，每个子立方体都由一个八叉树表示。图 4.2 显示了灵活八叉树的一个例子，在这个例子中，原

始八叉树的前四个子层被平坦化为一层，这意味着空间 W0 被划分为 16×16×16 = 4096 个子立方体(然后每个子立方体被递归地划分为 8 个更小的子立方体)。

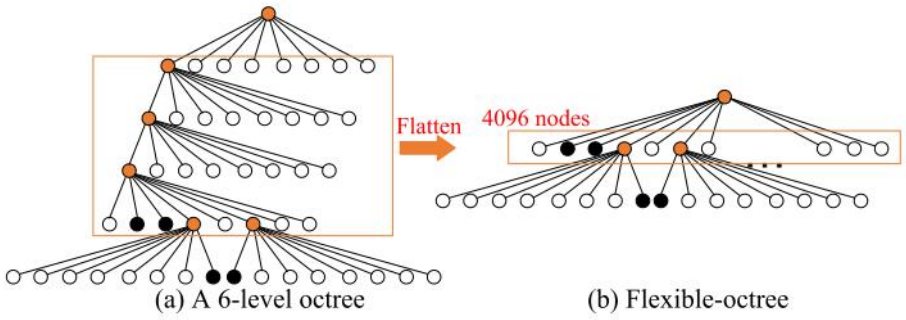

图 4.3 优化后的灵活八叉树

为了简化 PIM 逻辑设计，可以采用一种不同于 OctoMap[6]的节点编码方案。如前所述，每个节点都有三种状态:完全占据、部分占据和空着。因此，可以使用 2 位编码一个节点，编码方案如下所示:

00:空节点

01:满节点

10:未使用的节点

11:已占用的节点

利用新的编码方案，可以通过简单的逐位和运算来实现碰撞检测，如表 1 所示。两个节点的编码之间进行逐位和运算后，结果为 00 的结果为无碰撞，结果为 01 的结果为碰撞，如果结果为 11 则需要递归检查。这种位和操作可以在 DRAM 内部有效地进行，这将在下一小节中描述。事实上，如果一个八叉树被完全平展成一个没有子八叉树的灵活八叉树，那么可以只使用 1 位来编码每个节点。这样的架构和方法仍然适用，同时存储成本降低了一半。

表 4.1 利用与操作进行碰撞检测

AND	00	01	11	
00	00	00	00	collision-free
01	00	01	01	in collision
11	00	01	11	recursively check

4.3 DRAM 的内部优化

基于 Ambit 方法[7]开展。Ambit 是一款高效的内存加速器，支持 DRAM 内部的批量位操作。AND/OR 操作的主要思想是同时激活每个子数组的三行。图 4.3 解释了 Ambit 的与操作。

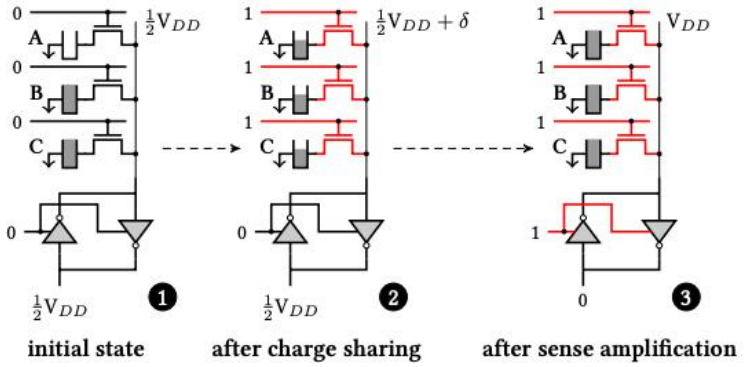

图 4.4 Ambit 中的与操作

首先准备晶体管 A、B 中的数据，将晶体管 C 初始化为 0，并将 BL 预充至 1 2VDD(图 6a)。然后三个 wl 同时被激活。在三个电池之间的电荷共享后，BL 电压会产生偏差。如果 A 和 B 都存储 1，则 BL 将被收取为 1 2VDD +Vd(图 6b)；否则，偏差为负，BL 电压变为 1 2VDD−Vd。最后启用 SA。根据第二步中的 BL 电压，将 BL 电压放大到 0 或 VDD(图 6c)。同时，三个电池将完全充电或放电。通过这样的方式，子数组中的所有 BLs 同时进行 AND 运算，可以实现内存内大规模并行处理。

为了支持 ambit 风格的内存和操作，需要对子数组架构进行修改，在其中增加了 3 行和一个额外的 WL 驱动程序，如图 7 所示。这两个操作数行首先被复制到附加行中的两行，然后在这两行之间执行按位和操作。WL 驱动程序在要的时候激活 3 个额外的行。

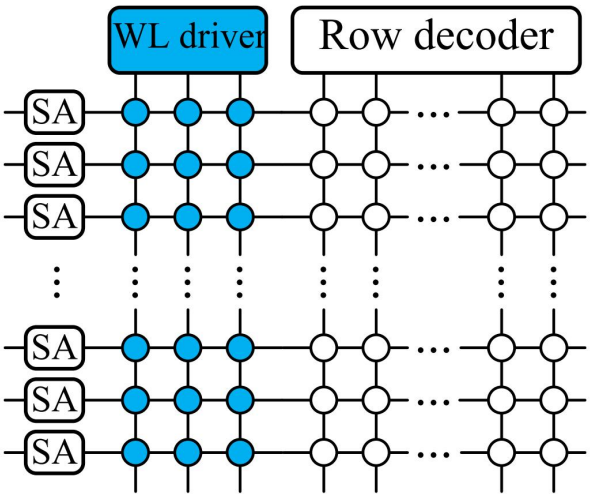

图 4.5 数组修改方式

如果使用不同的节点编码方案，则无法通过内存中的位和操作获得碰撞检测结果。相反，需要一些额外的数字逻辑，这会导致更高的开销。因此，新节点编码方案是利用简单的内存位和操作的关键。为了支持 DRAM 内部的批量位和操作，需要仔细分配数据。基本的要求是灵活八叉树的根的子代码必须是连续的，并且起始地址应该在 DRAM 行的头部对齐。如果 bank 宽度是 8192 位，那么有 4096 个子就可以满足要求。如果根节点的子节点数是 4096 的倍数，它也满足要求。更少的子节点需要填充数据来满足这一要求，会导致空间浪费。

4.3 PIM 逻辑设计

两个节点之间按位和运算产生一个 2 位结果，称之为单元结果。如果一个 DRAM 的宽度是 M 位，就会有 m2 单位的结果在每个 bank。这些单元结果被锁存在每个 bank 的行缓冲区中。仍然需要一些逻辑来生成最终的结果:如果两个灵活的八叉树是否发生碰撞，或者需要递归地检查它们。这是通过在每个 bank 中添加的 PIM 逻辑实现的。

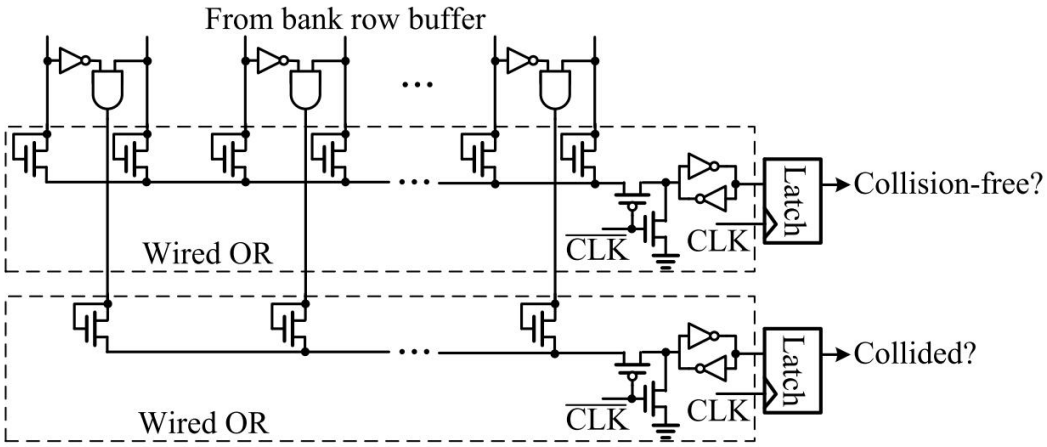

图 4.6 PIM 逻辑电路设计

在电路中，首先确定单元结果是否全部为 00，即两棵八叉树是否不发生碰撞。这可以很容易地通过一个 OR 逻辑来实现所有产生的位。另一种是看是否有单位结果为 01，即是否存在碰撞。这也很容易通过 OR 逻辑和一些附加逻辑来实现。为了节省面积和电力，使用动态逻辑风格设计的有线或逻辑来生成最终结果。对于两棵灵活八叉树之间的碰撞检测，DRAM 只向 CPU 发送两个 bit。因此，DRAM 和 CPU 之间的数据传输量是可以忽略的。

系统使用 PIM 体系结构来处理两个柔性八叉树的根节点的子节点。结果被发送到 CPU。如果需要递归检查，只需使用 CPU 或其他加速器来执行以下任务，不使用 PIM 体系结构进行后续递归，因为普通八叉树结构在柔性八叉树的较低级别上造成了较低的并行性。在内存中执行递归检查需要一个内存内核来控制递归流，并需要一个数据重新布局过程来提供足够的并行性，这两者都会带来非常高的开销。相反，由于执行递归检查的概率非常低，所以使用 CPU 或其他加速器执行递归检查不会产生太多开销。

5 测试与分析

通过设计 DRAM 中的 PIM 逻辑实现碰撞检测，能耗与传统的冯诺伊曼架构处理器相比有所降低。

表 5.1 能耗对比

Solution	Platform	Energy (mJ) of computation	Energy (mJ) of data transfer
Our accelerator	ASIC	1.282	0.847
Dadu-P [14]	ASIC	6.797	13.542

同时，分析在没有任何递归检查的情况下，柔性八叉树的第一级完成碰撞检测的概率。称之为快速完成率(RFR)。考虑四棵具有不同深度的八叉树，并在不同的层次上将它们压平。RFR 结果如图 5.1 所示。水平轴表示将八叉树扁平化以构建灵活八叉树的水平。如前所述，如果检测到两个挠性八叉树的根是碰撞的或无碰撞的，那么碰撞检测将在不递归检查的情况下完成。RFR 等于无碰撞和未碰撞的概率之和。从图 5.1 中可以看出，原始八叉树被压平的层次越多，得到的 RFR 越高。

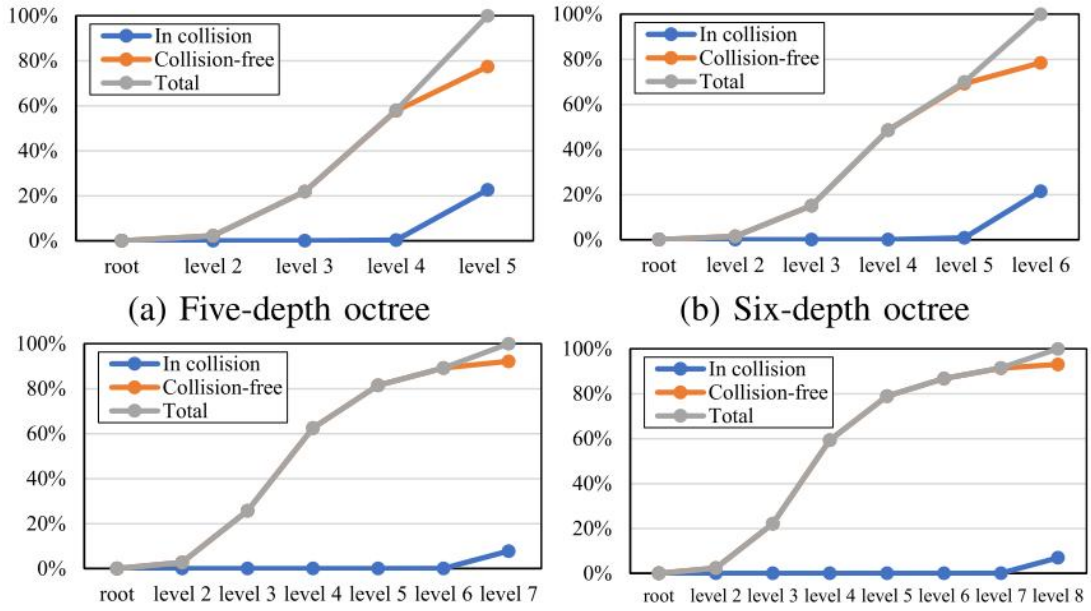

图 5.1 快速完成率对比

6 未来展望

如图 6.1 所示，2008 年以来，以忆阻器为代表的新型存储器件的出现，大幅促进了存内计算的研究和发展，在存算一体领域有一些初创企业，且已有产品上市，较成熟产品主要集中在低功耗、小算力、专用性领域。而存算一体加速器凭借其并行性高、能效比高的特点，在低制程下往往能表现出不亚于高制程通用 GPU 的能效比，在神经网络加速器领域具有自身独特的优势。

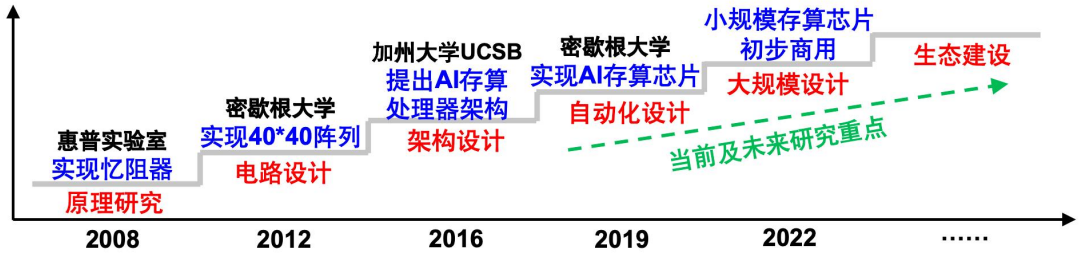

图 6.1 存内计算架构发展历史

目前，世界各个头部实验室与企业都在尝试将目前嵌入式的存算一体器件扩大规模，使其成为极具能效比的云端大算力平台，从专用存算机到通用大算力存算机的过程。个人认为，存算一体从专用到通用的重点内容是生态建设，包括存算一体标准化指令集的建设、存算一体编译器的开发以及存算一体操作系统的建设。目前，对存算一体指令集、编译器的研究工作很多，但仍不成熟，但由于存算一体的复杂性高、成熟度地、硬件集成度要求高，相关操作系统的研究工作尚未开展。

目前在存算一体加速器领域，清华大学集成电路学院的 STELLAR 加速器、复旦大学陈迟晓团队的 COMB-MCM 边缘端加速器、中科院计算所陈晓明团队的 Dadu-cd 机器人碰撞检测加速器，均为该领域前沿研究成果。

存算一体加速器的大算力、通用性发展也非常符合我国“东数西算”、“算力网”等的国家算力迭代战略发展需求，也许在存算一体编译技术领域有了长足发展后，有望建成存算一体数据中心，充分发挥存算一体芯片既能存储又能计算、高能效比的优势。

参考文献

[1] Petrenko S, Petrenko S. Limitations of von neumann architecture[J]. Big Data Technologies for Monitoring of Computer Security: A Case Study of the Russian Federation, 2018: 115-173.

[2] Yang Y, Chen X, Han Y. Dadu-CD: Fast and efficient processing-in-memory accelerator for collision detection[C]//2020 57th ACM/IEEE Design Automation Conference (DAC). IEEE, 2020: 1-6.(主要阅读)

[3] Zhu H, Jiao B, Zhang J, et al. COMB-MCM: Computing-on-memory-boundary NN processor with bipolar bitwise sparsity optimization for scalable multi-chiplet-module edge machine learning[C]//2022 IEEE International Solid-State Circuits Conference (ISSCC). IEEE, 2022, 65: 1-3.

[4] Zhang W, Yao P, Gao B, et al. Edge learning using a fully integrated neuro-inspired memristor chip[J]. Science, 2023, 381(6663): 1205-1211.

[5] Han Y, Yang Y, Chen X, et al. DaDu series: fast and efficient robot accelerators[C]//Proceedings of the 39th International Conference on Computer-Aided Design. 2020: 1-8.

[6] Hornung A, Wurm K M, Bennewitz M, et al. OctoMap: An efficient probabilistic 3D mapping framework based on octrees[J]. Autonomous robots, 2013, 34: 189-206.

[7] Seshadri V, Lee D, Mullins T, et al. Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology[C]//Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture. 2017: 273-287.