中国科学: 信息科学 2022 年 第 52 卷 第 3 期: 377–398
SCIENTIA SINICA Informationis
《中国科学》杂志社
SCIENCE CHINA PRESS
评述

计算体系架构研究综述与思考

高彦钊*, 邬江兴, 刘勤让, 沈剑良, 宋克, 张帆

信息工程大学信息技术研究所, 郑州 450003
* 通信作者. E-mail: buaagaoyz@sina.com

收稿日期: 2021–05–13; 修回日期: 2021–06–21; 接受日期: 2021–07–07; 网络出版日期: 2022–03–04

国家科技重大专项核高基项目 (批准号: 2016ZX01012101) 和国家科技重大专项核高基项目 (批准号: 2017ZX01030301) 资助

摘要 随着摩尔定律 (Moore's law) 与迪纳德 (Dennard) 缩放定律逐步走向终结, 依靠集成电路制程工艺的进步提升计算系统性能与效能越来越困难, 计算体系架构的演进成为了未来计算系统发展的重要技术途径. 本文首先从应用适应性、计算驱动方式、系统重心变化、计算核心构成, 以及计算逻辑使用等不同的角度回顾了体系架构的发展历程, 总结了不同体系架构的优缺点; 然后着重分析了在人工智能、大数据等应用飞速发展的条件下未来计算系统的能力需求特征; 最后提出了软件定义计算体系架构, 并梳理了其重点研究内容与关键技术, 为未来计算体系架构的发展提供了一条可行的技术途径.

关键词 体系架构, 软件定义计算, 领域专用架构, 异构计算, 可重构计算

1 引言

1946 年世界第 1 台电子数值积分计算机 ENIAC 的诞生标志着人类进入计算自动化革命的新阶段. 在几十年的发展历程中, 自动化计算乃至智能化计算取得了长足的发展, 有力地推动了人类社会的进步. 计算技术的主要推动力主要包括两个方面: 其一是制程工艺的进步, 自 1959 年现代集成电路发明以来, 芯片晶体管集成度按照摩尔定律 (Moore's law) 预测每 18 个月翻一番, 带来了计算性能的快速提升; 其二是体系架构的演进, 从冯 · 诺依曼 (von Neumann) 架构开始, 根据应用需求的变化以及技术指标的侧重提出了多种不同类型的计算体系架构, 在 "基因" 层面助力计算系统获得计算性能、效能或灵活性等方面的巨大收益.

随着摩尔定律与迪纳德缩放 (Dennard scaling) 定律逐步逼近物理极限 [1], 一方面晶体管集成密度的提升越来越困难, 另一方面晶体管集成密度的提升带来了功耗墙问题, 依靠制程工艺进步提升计算系统性能与效能的途径已经难以为继. 因此, 除了一些机构继续在提升芯片制程工艺方面深耕外, 不论学术界还是产业界, 越来越多的研究人员将目光关注点锁定在计算体系架构革新这一领域中, 而近

引用格式: 高彦钊, 邬江兴, 刘勤让, 等. 计算体系架构研究综述与思考. 中国科学: 信息科学, 2022, 52: 377–398, doi: 10.1360/SSI-2021-0163
Gao Y Z, Wu J X, Liu Q R, et al. Review and thoughts on the development of computing architecture (in Chinese). Sci Sin Inform, 2022, 52: 377–398, doi: 10.1360/SSI-2021-0163

年来由计算体系架构创新设计带来的计算系统功能性能指标提升为计算系统设计与应用注入了新的活力.

体系架构是指计算、存储以及互连等一组部件的组织形式与使用方法 [2], 是人工复杂系统研究的核心范畴 [3], 不仅决定着计算系统的功能与性能, 还决定着计算系统的效能与安全. 国内外研究者们在自动化计算技术发展之初即注意到体系架构对计算系统的影响, 因此人们对体系架构的研究与探索从未停滞. 几十年来, 围绕高速、高效、灵活、安全等目标体系架构不断向前发展与演进, 特别是近年来随着人工智能、大数据、云计算, 以及物联网等技术的快速发展与广泛应用, 对计算系统提出了越来越高的要求, 催生着新颖的计算体系架构不断涌现, 其内涵与外延得到了极大的丰富.

针对体系架构的发展脉络与未来发展方向问题, 本文第 2 节从应用适应性、系统重心变化、计算驱动方式、计算核心构成, 以及计算逻辑使用方法等多个方面对体系架构的发展历程进行了梳理, 总结了其各自的优缺点与适用范围; 第 3 节在应用需求、计算需求, 以及系统应用等 3 个方面分析了未来计算系统的能力需求; 第 4 节综合未来计算系统需求与现有计算架构的问题, 提出了软件定义计算体系架构的概念、特点、层次化结构及其核心研究内容与关键技术点; 第 5 节对本文进行了总结.

2 体系架构的发展历程

2.1 从应用适应性角度看体系架构

应用适应性是在维持原有体系架构完整不变的条件下, 以微小的经济与时间代价实现对不同应用需求适应能力的一种描述 [4]. 从 19 世纪初巴贝奇 (Babbage) 设计世界上第 1 台机械式计算机开始, 从应用适应性角度看, 受应用需求变化、设计思想限制, 以及工艺水平发展等客观条件的影响, 计算体系架构大致经历了专用计算到通用计算再到通专并行, 并逐步发展到领域专用计算这 4 个阶段.

2.1.1 专用计算

在计算机 ENIAC 出现之前, 人类在自动化计算方面进行了多次尝试. 受限于当时的技术条件, 自动化计算机器往往是面向专用功能而设计的, 例如为存储计算资料而研制的电动制表机, 为解算复杂数学方程而设计的电子计算机 ABC 等. 1946 年 ENIAC 诞生, 专门用于计算弹道, 但是其运算方式却与今天的计算机有很大的不同: 首先, ENIAC 不具备今天的 “软件” 功能, 其运算模式必须事先确定且通过硬连线实现硬件重组; 其次, ENIAC 不具备存储功能, 数据在完成运算输出后就不复存在, 不能为其他运算所用.

2.1.2 通用计算

1945 年, 冯 · 诺依曼与戈德斯坦 (Goldstein) 等联名发表了计算机史上著名的 “101 页报告”, 提出了冯 · 诺依曼计算架构. 如图 1 所示, 冯 · 诺依曼架构是存储 – 程序通用电子计算机的设计模型, 包括存储器、算术逻辑单元、控制单元、输入设备, 以及输出设备等, 将 CPU 与内存分开, 通过设计不同的指令集, 可以使计算机执行多种不同的功能, 具有非常高的灵活性, 改变了早期计算机器程序固定、功能单一的缺点, 具有 “能够计算一切可计算问题” 的高度灵活性, 对后世产生了极其深远的影响.

冯 · 诺依曼体系架构采用指令与数据统一编址、一体存储, 并共用一组总线传输的方式. 文献 [5] 将冯 · 诺依曼体系架构进一步抽象, 形成了通用计算体系架构, 如图 2 所示. 该体系架构的主要特点是系统的运行受到程序的控制, 其工作原理与冯 · 诺依曼体系架构基本一致: 控制器内含控制码存储器, 一方面受外部输入的应用程序驱动, 另一方面实时接收数据通路回写的状态, 在外部时钟的控制

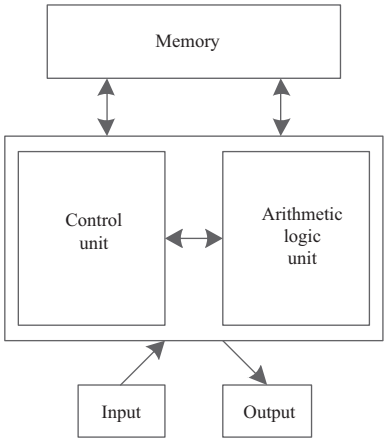

图 1 冯 · 诺依曼架构

Figure 1 von Neumann architecture

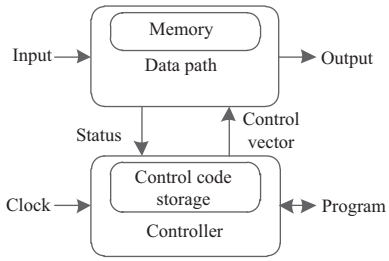

图 2 通用计算体系架构

Figure 2 General-purpose computing architecture

下, 根据程序要求与状态变化依次向数据通路发送相应的控制向量驱动计算进程; 而数据通路包含存储器, 通过接收控制向量, 决定取用什么样的数据 (包括外部输入数据与存储数据), 完成什么样的计算, 并将计算结果与系统状态输出.

随着信息化革命的不断深入与智能化革命的快速崛起, 多样化的应用需求、个性化的服务需求, 以及爆炸性的数据增长对计算系统的性能提出了越来越高的要求, 促使人们在冯 · 诺依曼体系架构及其演进形式的框架内, 对通用计算体系架构采用了多种方式以提升其计算性能. 概括来说主要包含两个方面: 一方面通过制程工艺的不断提升增加单位面积的晶体管数量. 事实上自 1959 年现代集成电路发明以来, 在几十年的发展历程中制程工艺基本按照 0.714 的代际比例不断进步, 集成度的升高使得中央处理器 (CPU) 的时钟频率快速提升, 同时也带来了成本的快速下降, 不仅促成了处理器按照摩尔定律快速发展, 其性能每隔两年会提升一倍, 同时也促成了 20 世纪 “为大众计算”, 让计算走入千家万户的空前繁荣局面; 另一方面则是体系架构的不断演进, 其中在宏观层面, 为了解决在高速、实时处理时造成的总线拥挤, 出现了将指令和数据存储在不同空间中并采用指令与数据并行存取方式的哈佛架构 [6]; 在微架构层面, 多级流水线 [7]、并行多线程 [8], 以及多发射与乱序执行 [9] 等新技术不断应用, 改善了通用计算指令执行的并行性与流水化, 提升了计算执行效率; 在指令集层面, 从适应于最早分立元件硬件系统的少量指令集合到伴随着现代集成电路发展而形成的复杂指令集、精简指令集, 以及超长指令字 [10], 再到专门针对某类应用需求特别定制的指令集等, 指令集的体系化发展不仅带来了计算效率和应用适应性的大幅提升, 也在一定程度上缓解了通用计算的功耗问题; 在计算核层面, 2005 年

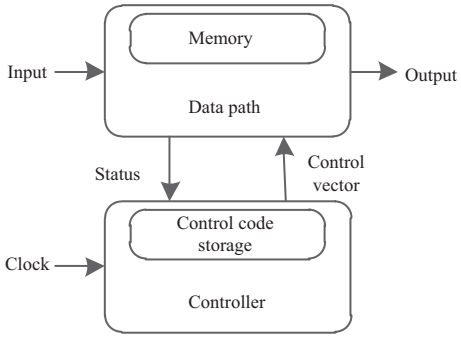

图 3　专用集成电路体系架构

Figure 3　Application-specific integrated circuit architecture

以后当晶体管尺寸缩小带来的时钟频率提升难以为继时, 计算核数量由单核逐步增长为双核、多核乃至众核, 依靠多核并行与协同为通用计算的发展注入了新的活力.

事实上, 通用计算基于制程工艺进步与基于计算架构演进的两种发展途径是齐头并进、交替应用的. 英特尔的处理器研发自 2008 年开始严格遵循 Tick-Tock 研发策略: 如果新一代 CPU 对应 Tick, 那就意味着相对于前一代采用更高的制程工艺提升 CPU 性能; 如果新一代 CPU 对应 Tock, 那就意味着相对于前一代将会进行处理器微架构升级.

2.1.3　通专并行

1959 年, 在德州仪器公司工作的杰克 · 基尔比 (Jack Kilby) 与罗伯特 · 诺伊斯 (Robert Norton Noyce) 发明了现代集成电路. 得益于现代集成电路的飞速发展, 一方面通用处理器的计算性能每两年翻一倍, 另一方面专用集成电路 (ASIC) 在 20 世纪 90 年代迅速崛起. 专用集成电路是面向某种特定应用专门定制化设计的集成电路, 已经在多个领域广泛应用 [11, 12], 在当前与通用处理器共同形成了通用与专用并行的算力格局. 专用集成电路体系架构与通用计算体系架构相比, 最大的差别在于系统的行为是确定的, 如图 3 所示 [5]. 其控制器中的控制向量根据专门的应用需求定制化且无冗余设计, 在计算任务执行过程中, 根据数据通路中回送的数据或计算状态, 决定下一个时刻向数据通路发送控制向量的种类. 与通用处理器相比, 专用集成电路控制向量定制且无冗余节约了计算资源, 计算过程并行且流水化提高了计算效率, 资源使用直接且更接近底层 I/O, 因此具有体积小、功耗低、计算性能高、计算效率高等优势.

2.1.4　领域专用计算

随着近年来计算数据爆炸增长、应用需求多样发展、便携设备广泛应用, 以及工艺发展明显放缓, 尤其是在人工智能处于大爆发时期, 大量的算法不断涌现, 远没有到算法平稳期, 对计算系统的性能、效能、灵活性, 以及可靠性等提出了越来越高的要求. 面对这样的困境, 除了一些机构继续在提升芯片制程工艺方面深耕外, 而许多公司与研究机构则跳出通用计算体系架构与专用计算体系架构, 探索新的发展方向. 2009 年, 在通用计算与专用计算尚处于高速发展期时, 邬江兴院士及其研究团队已经洞察到了未来计算系统对性能、效能、灵活性, 以及可靠性等各方面的综合需求是通用计算与专用计算体系架构远远无法满足的, 率先开展了新概念高效能计算体系架构的研究, 提出了拟态计算的概念 [13], 开辟了领域专用软硬件协同计算技术方向, 具备原理普适性、技术泛在性和军民两用性特征, 具有结构创新内生增益和非对称战略意义, 并于 2013 年成功研制世界首台拟态计算机, 入选当年的中国十

大科技进展, 随后拟态计算思想在各个领域得到广泛应用 [14~19]. 2017 年, DARPA 在电子复兴计划新增新项目中包括了领域专用片上系统 (domain specific system on chip, DSSoC) [20] 和软件定义硬件 (software defined hardware, SDH) [21] 两个项目, 旨在建立一种软硬件解耦的架构, 通过软件定义的方式实现硬件功能甚至芯片功能的重构与扩展. 在 ISCA2018 大会上, 图灵奖得主 John Hennessy 与 David Patterson 发表 “A new golden age for computer architecture” 演讲, 指出在摩尔定律走向终点的同时, 体系架构正在闪耀新的活力, 领域专用架构将会兴起 [22]. 清华大学针对人工智能算法研制了可重构智能计算芯片 —— Thinker [23], 支持多种深度学习算法的高效重构, 功耗极低, 效能极高, 能够广泛部署在移动设备、可穿戴设备, 以及物联网应用中. 2021 年美国半导体研究协会发布未来十年半导体研究规划 [24], 更明确指出, 到 2030 年, 我们预计将进入一个领域专用计算时代 (每个问题类可能一个系统类型). 因此, 作为未来重要发展方向, 领域专用计算已经引起了学术界与工程界的广泛关注.

2.1.5 小结

通用计算体系架构为了适应广泛的应用需求, 采用了从应用任务到计算逻辑之间多层虚拟化的实现方式, 导致了其晶体管利用效率低下, 在制程工艺举步维艰的情况下, 已经无法从增加晶体管数量这一途径中获取明显的收益, 因此其性能, 尤其是效能的提升已经受到了严峻的挑战; 专用计算体系架构虽然针对特定的算法具有明显的性能与效能优势, 但其内置算法是固定的, 灵活性非常差, 一旦算法变化就可能无法使用, 而重复设计的时间成本与经济成本非常高; 而领域专用计算能够充分利用领域内不同应用在计算与存储的共同特点, 对计算逻辑进行半定制与小冗余设计, 能够实现计算性能、效能与灵活性的有机平衡与综合提升, 是未来计算体系架构的重要发展方向.

2.2 从系统重心变化看体系架构

组件呈现形式及组件互连关系构成了体系架构的核心部分. 虽然概括地说组件主要包括存储部件、计算核心、互连结构, 以及外围的输入与输出设备等, 但是在计算设备发展的不同历史时期, 系统设计与运行的中心是有所变化的, 大致可分为以处理器为重心、以存储器为重心, 以及以总线为重心 3 个阶段.

2.2.1 以处理器为重心

处理器是计算系统的核心计算部件. 在处理器发展初期, 其计算性能有限, 而存储器容量较小, 两者的发展相对均衡, 加上计算系统本身对信息存储要求较低, 而且人们的关注重点更多地在于提升处理器计算效率, 因此, 计算系统体系架构的设计是以处理器为重心的. 1971 年, 由英特尔 (Intel) 公司研发的世界上首款微处理器诞生, 运算器与控制器合二为一, 处理器的发展进入了快车道, 数据位宽从最早的 4 位逐步发展到 32 位乃至 64 位, 时钟频率从最早的 4.77 MHz 逐步发展到几百 MHz 乃至几 GHz. 从 20 世纪 80 年代开始, 存储器与处理器之间从性能、发展速度等方面就开始出现了较大的差距. 但是在差距还不足够大的情况下, 人们虽然采用了一系列的措施弥补两者之间的差距, 如设计更大的片上 Cache、更宽更快的片外存储带宽等, 但是此时研究的着力点仍然在处理器性能提升方面. 在这个阶段内, 计算系统仍然以处理器为重心进行设计开发. 以处理器为重心的体系架构如图 4 所示, 是一种非常典型的冯 · 诺依曼体系架构, 包括单个中央处理器 (CPU)、存储器, 以及输入/输出设备等, 数据的传输与处理均要通过处理器完成.

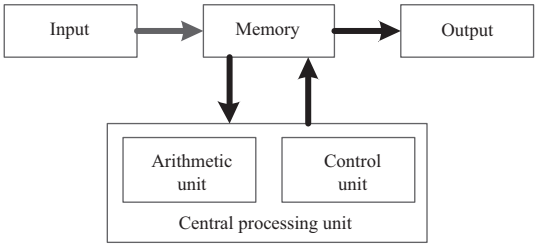

图 4　以处理器为重心的体系架构

Figure 4　Processor-centric architecture

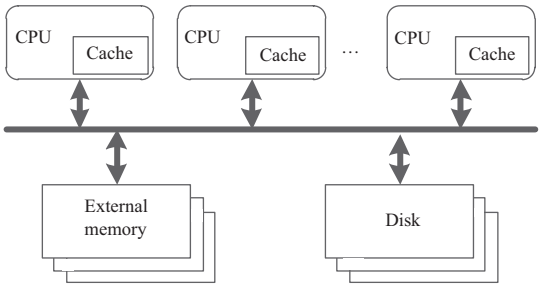

图 5　以存储器为重心的体系架构

Figure 5　Memory-centric architecture

2.2.2　以存储器为重心

根据 Amdahl 定律 [25], 具有高性价比特性的计算系统其带宽应当是平衡的. 但是在过去几十年中, 处理器基本遵照摩尔定律的预测快速发展, 即每过 18 个月芯片集成晶体管数目翻一番, 而存储器每年的速度则仅为 7%. 究其原因, 两者设计目标有明显的差别: 存储器是以容量最大化、成本最小化为设计目标, 而处理器则以性能最大化为设计目标 [26], 因此工业界也划分为了两个明显的阵营, 从设计方法、设计目标, 以及生产工艺等方面有着明显的差别. 自 20 世纪 80 年代开始, 存储器与处理器之间的发展速度差异仍然在不断加剧. 当数据访存速度难以满足计算需求时, 计算系统的发展面临着越来越严重的 "存储墙" 问题. 虽然人们采取了多种手段提升数据访存速度, 包括提升带宽、增加缓存、设计层次化存储结构, 以及改变数据存取方式等, 以求缓解算存比不平衡的问题, 但是至今尚无根本的解决办法. 因此, 体系架构设计重心逐渐由处理器转移到存储器, 提出了近存储计算 [27]、存内计算 [28] 等概念, 以存储器为重心的体系架构逐渐成了主流.

如图 5 所示, 以存储器为重心的体系架构一般由多个处理器组成, 处理器围绕存储器便于就近完成数据访存操作. 另外为了提升数据访存速度, 设计了由寄存器、高速缓存、主存储器与外部存储器等多级存储结构 [3].

2.2.3　以互联为重心

随着数据密集程度不断提升, 对计算系统的性能需求快速增长, 单计算系统已经无法满足大数据量的应用需求, 往往需要多套计算设备组合在一起形成庞大的计算系统. 特别是随着超级计算机、云计算, 以及分布式计算等发展, 需要借助总线结构将多个处理机与存储系统结合起来, 通过控制系统的调度管理解决不同处理系统与存储系统之间大规模并发需求, 完成大型复杂计算任务的协同计算. 因此, 在更高的层面上以总线为重心的体系结构应运而生. 总线结构是实现计算系统数据与指令汇聚与分发的设计重心, 先后经历了共享总线与交换总线两个阶段. 以总线为重心的体系架构如图 6 所示.

另一方面, 随着集成电路的发展, 20 世纪 90 年代中期出现了将多个具有特定功能的集成电路组合在单芯片上的系统或产品, 即片上系统 (SoC), 已经成为集成电路产业未来的重要发展方向. 随着片上处理核心的数量增多、规模增大, 片内处理核心及其与片外其他部件之间的高效低耗信息传输与交互对于提高片上系统性能具有重要意义. 因此, 以互联为重心的计算体系架构也下沉到单个芯片内部, 片上网络 (network on chip, NoC) 近年来已经成为了研究的热点, 主要包括体系结构 [29,30]、互连拓扑 [31,32]、路由方法 [33,34]、流控机制 [35,36], 以及容错机制 [37,38] 等关键技术.

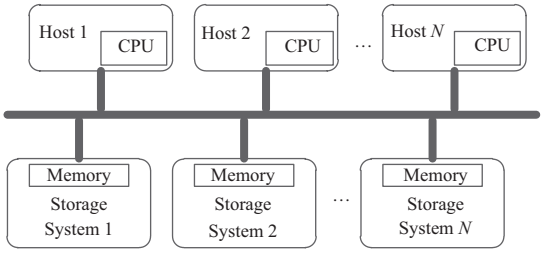

图 6 以总线为重心的体系架构

Figure 6 Interconnection-centric architecture

2.2.4 小结

计算架构重心的变化可以看出, 在不同的时期, 受应用需求和技术水平的影响, 虽然体系架构的设计有不同的着重点, 但是其总体目标旨在克服计算系统的短板问题, 追求系统的平衡性. 而体系架构重心从计算与存储部件变化为部件之间的连接关系也说明了当前体系架构在朝着平面化与去中心化方向发展, 即所有计算部件与存储部件地位相同, 而如何将这些部件有效连接并充分利用起来是体系架构设计的关键问题. 当前, 在计算系统朝着多种功能一体化、大众服务个性化、新业务高效部署方向发展, 特别是在云计算与边缘计算的快速崛起的情况下, 对计算系统灵活性、高效性、开放性, 以及可扩展性提出了越来越高的要求. 受此影响, 计算系统正在由以总线为重心向以软件定义互连结构为重心进一步演进. 新一代软件定义体系架构 [3] 以模块化、标准化的软件定义节点 (包括计算节点与存储节点) 为基础, 以软件定义互连结构实现对软件定义节点之间的层次化组织, 能够根据应用需求改变计算结构, 最优化匹配应用计算需求, 在体系架构层面实现系统性能、效能、灵活性, 以及可靠性的综合平衡与同步提升, 是未来体系架构重要的发展方向.

2.3 从计算驱动方式看体系架构

计算驱动方式是指在计算过程中计算操作能够被执行的触发条件类型. 从通用计算广泛应用开始, 对应于不同的计算体系架构, 计算驱动方式主要包括指令流驱动、数据流驱动、配置流与数据流共同驱动, 以及事件驱动.

2.3.1 指令流驱动

冯 · 诺依曼体系架构的一个重要贡献是提出了指令的概念. 指令反映了计算系统所具备的基本计算功能 [10], 也是应用软件在硬件逻辑上运行的基本单元. 指令流驱动方式是指在编译系统支撑下将应用软件转化为有序指令序列, 根据系统运行回写的状态下发相应的指令, 并在指令控制下完成数据读取、计算与回写的计算驱动方式. 本质上说, 指令流驱动是一种依据指令顺序分时复用计算逻辑的时域串行计算方式. 通用计算体系架构与指令流驱动相结合使得通用处理器件可以计算一切可计算问题, 并由此衍生出诸多高级程序语言.

指令集的发展是一个由少到多再逐步精简, 由简单到复杂再逐步回归简单的过程 [10]. 在 20 世纪 50 到 60 年代, 一方面计算机硬件结构比较简单, 另一方面应用任务及其对计算性能的要求相对较低, 因此指令数量相对较少; 60 年代中期, 现代集成电路的出现为计算机的发展注入了强大的推动力, 不仅计算性能快速提升, 而且其系统规模、成本, 以及功耗等快速下降, 伴随而来的是指令系统的日益复杂化; 到 70 年代, 出现了具有里程碑意义的复杂指令集 [39].

复杂指令集是在集成电路技术的快速发展的条件下, 为提升软件编程便捷性与程序运行速度, 通过不断增加具备实现复杂功能能力的指令, 并匹配以多样化编址/寻址方式而逐步形成的指令集合, 其主要特点是指令数量多且长度不定, 多时钟周期执行时长, 不同指令使用频率差别大, 支持多种寻址方式, 以及采用微程序控制技术等 [10]. 发展至今, 复杂指令集面临着多重问题: 虽然能够减小高级语言与机器指令之间的语义差距, 但是必须以增加硬件复杂度与多周期执行为代价; 虽然采用微程序提升了代码密度, 但是该策略与超大规模集成电路中更适合采用硬布线控制逻辑的现实相悖; 为了保持向后兼容性, 复杂指令集保留了许多已经过时的定义, 导致指令集冗余臃肿, 应用开发门槛提高. 但是复杂指令集仍然具有较强的生命力, 早期的 CPU 全部采用复杂指令集, 即使在对处理器功耗越来越敏感的当前, 其仍然在基于 X86 架构的通用处理器上广泛应用.

针对复杂指令集冗余臃肿、结构复杂等弊端问题, 研究者秉承简单设计哲学, 在仅保留少量使用频率高的指令与部分支持高级语言与操作系统的必要指令的基础上, 配合统一格式指令译码、简单寻址方式等技术提出了精简指令集. 这与通过增加硬件复杂度以适应指令复杂性并提升计算性能的复杂指令集设计思路具有本质的区别, 即将设计复杂度从硬件层面上移至编译系统, 以软件复杂度提升换取硬件复杂度与计算功耗的下降. 到目前为止 RISC 已经迭代发展了 5 代, 而第 5 代 RISC-V 尤其受到了人们的关注. 第 5 代 RISC-V 是第 1 个能够根据具体场景选择合适指令集的开放性指令集架构 [40], 仅以 40 多条基础指令实现面向不同应用的定制设计, 不仅是一种架构比较简单且具有完整工具链的指令集, 而且其完全开源的特点也促进了其在各个领域的广泛应用, 已经受到了学术界与工业界的广泛关注 [41~43].

2.3.2 数据流驱动

数据流驱动是指采用数据流图描述计算任务, 以计算所需数据就绪且有效为触发条件, 相应运算操作开始执行的计算驱动方式. 与指令流驱动不同, 数据流驱动执行顺序不需预先设定, 而是在程序运行时根据计算进程、数据间的依赖性、操作数的有效性, 以及计算系统状态动态确定, 最早由麻省理工学院 (Massachusetts Institute of Technology) 的 Dennis 提出 [44, 45]. 本质上说, 数据流驱动是一种计算顺序取决于数据相互依赖关系及操作数有效性的空域并行计算方式. 因为计算程序转换成数据流图存在困难, 没有数据数据共享机制导致存储空间浪费以及在执行过程不确定时难以进行程序编写与调试, 所以纯粹的数据流驱动计算机没有产业化, 但是数据流驱动计算思想在指令流驱动计算优化、流处理器设计、大数据与智能计算应用等各个层面获得了广泛应用 [46].

在指令流驱动计算优化层面, 指令流水线设计方法借鉴了数据流驱动计算思想, 通过对指令流片段进行数据依赖性分析以及分支预测等操作, 然后根据预测的数据值是否就绪作为指令执行的触发条件; 指令乱序执行技术在译码时以不同寄存器存储微指令, 并对数据总线输出实施监控, 等待操作数到位后立即执行相应的微指令; 通过建立重排序缓冲区, 单个指令周期内不同微指令能够在不同处理单元上并行执行. 这些技术均借操作数有效且就绪时, 立即执行计算的数据流驱动计算思想.

在流处理器设计方面, 具有计算访存比高、计算规律固定、可并行处理, 以及局部数据重用率高等特点的应用需求为数据流计算思想的发展提供了新契机. 将数据表示为连续排列的记录格式, 并按照一定的方向持续流动以完成计算的方式称为流处理方式. 文献 [47] 将数据流传递过程与计算过程分离, 分别通过 Stream 与 Kernel 模型进行描述, 然后将两者组合起来形成了 Stream-Kernel 编程模型, 这与数据流驱动计算思想是一致的. 基于 Stream-Kernel 两级编程模型, 流处理器对硬件结构进行了专门设计, 实现了数据访问、指令组织, 以及指令操作的解耦. 典型的代表是 Imageine [48], RAW [49], 以及 Merrimac [50] 等, 其中 Imageine 采用流寄存器文件代替 Cache 实现数据缓冲存储, 其原型系统

在 250 MHz 主频下计算能力能够达到 10 Gflops. 文献 [51] 提出了 Godelet 编程模型, 提升了数据流图描述的粒度, 引入了层次化并行思想, 增加了局部数据共享性. 文献 [52] 基于 Codelet 模型实现了 DARTS 系统, 使用人员能够基于该系统使用高级语言在通用硬件上进行编程, 将 Codelet 模型中的线性程序与循环等要素准确表达出来. 为提高 Codelet 在执行迭代任务时的资源利用率, 文献 [53] 面向科学应用, 采用迭代与主循环分离的方式设计了数据流加速器, 能够在多个层面提高并行度. GPU 是结合控制流驱动与数据流驱动方式的处理器, 从 2006 年开始逐步实现了从 CPU 附属的专用图形处理器向通用计算器件的转型, 一方面继承并发展了 Stream-Kernel 模型中的多级编程方式, 另一方面通过堆叠大量可并行的流处理器作为其众核计算单元, 因此既具有良好的可编程性, 又能够大幅提升计算性能.

在云计算大数据方面, 谷歌 (Google) 基于数据流驱动计算思想提出了 MapReduce 框架 [54], 通过用户自定义的 Map 和 Reduce 操作实现实时数据大规模并行计算和容灾备份. MapReduce 虽然擅长做全量数据的离线处理, 但是显然并不具备足够的实时性, 因此 Twitter 设计了一种开源的、具有分布式实时与容错计算特点的计算框架 —— Storm [55], 在该框架中数据以流的形式依次处理生成结果. Spark 框架 [56] 同样集成了数据流驱动的思想, 支持在内存中对数据进行迭代计算, 极大提高了并行计算的速度. Flink [57] 是一个拥有增量迭代计算的流式数据处理框架, 对迭代计算和流式计算的支持较强.

在智能算法加速方面, 一方面, 在对网络结构描述与编程中, 研究者通过构建数据流图描述人工神经网络的各个层次的算子及其互连关系; 另一方面, 在硬件加速器结构设计中, 基于人工神经网络结构特点与计算特点, 在不同层级或颗粒度上能够开展并行化、流水化设计以提升计算效率, 包括在层级计算流水线设计、特征图级并行通道设计、窗口级局部数据共用, 以及操作级累加树构建等. 这些均体现了数据流驱动计算思想 [46].

2.3.3 配置流与数据流共同驱动

配置流与数据流共同驱动的计算方式是与可重构计算技术相伴相生的, 尤其是在 20 世纪 90 年代粗粒度可重构技术兴起之后, 迅速得到了发展. 可重构计算系统的设计出发点是以计算逻辑根据应用而变化的方式实现通用计算架构的灵活性与专用计算架构的性能与效能相结合, 在灵活性、性能, 以及效能之间做出合理的折中, 后续将做出详细论述. 可重构计算处理器体系架构如图 7 所示 [5], 其中图 7(a) 描述了可重构计算处理器硬件架构, 主要包括可重构数据通路 (RCD) 与基于可编程有限状态机 (PFSM) 的可重构控制通路 (RCC), 两者是分离的; 图 7(b) 描述了对应的编译器架构, 实现从高级编程语言到配置信息的转化.

图 7(b) 可重构计算处理器编译器中, 高级编程语言实现对不同应用算法的直接描述, 经编译器后生成与可重构处理器硬件架构相匹配的配置信息; 而从图 7(a) 可重构计算处理器硬件架构中可以看出, 整个处理器在外部时钟的激励下, 完成计算任务需要两步: (1) 通过配置信息完成对 RCC 与 RCD 的配置, 即实现对实现系统有限状态机与计算、存储与互连资源的重构, 使之形成类似于图 3 中专用集成电路的体系架构; (2) 在数据流的驱动下, 完成应用任务的计算. 这种配置流与数据流共同驱动的计算方式与指令流驱动、数据流驱动的计算方式相比, 既保留了一定的灵活性, 实现了对一定范围内不同应用任务的兼容性, 又实现了计算架构随应用任务的计算特点而灵活变化, 达到接近专用集成电路的性能与功耗 [58].

但是, 当前学术界对 CGRA 的架构和编译优化模型比较分散, 且研究平台需要一定积累, 导致研究门槛较高. 文献 [59] 介绍了一种开源 CGRA 框架, 该模型提炼多种 CGRA 架构的共性特征, 将架构模型、映射编译和物理实现解耦, 支持通过基于 LLVM 的编译器生成配置流, 使得可重构计算在灵

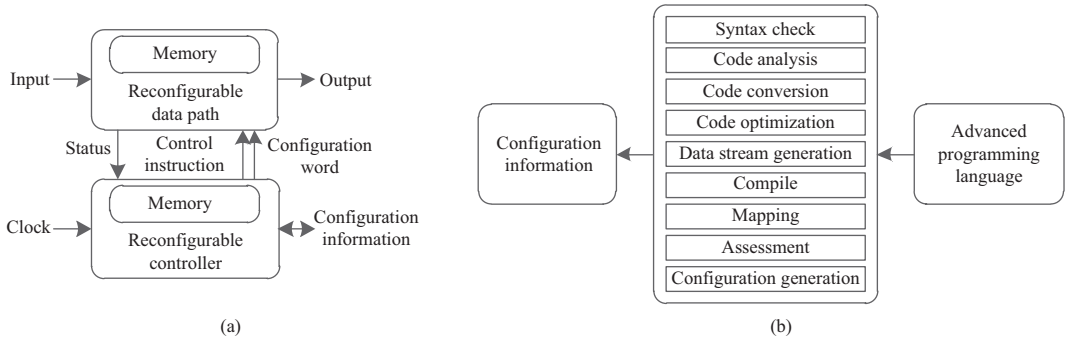

图 7　可重构计算处理器体系架构

Figure 7 Reconfigurable computing processor architecture. (a) Reconfigurable computing processor hardware architecture; (b) reconfigurable computing processor compiler

活性和开发难度上更具优势. 从以上研究可以看出, 以可重构计算为代表的配置流驱动已经成为一种认可的、高效的计算驱动方式.

2.3.4　事件驱动

事件驱动的计算方式与上述驱动方式有本质的不同, 是以脉冲神经网络 (SNN) 为代表的神经拟态计算采用的计算驱动方式, 也是对人脑信息处理方式更为贴切的模拟. 生物神经网络主要通过神经元发放与接收脉冲 (包括脉冲时间与脉冲频率等) 实现神经元之间的信息交互. 每一次脉冲交互都可以看作是一个事件, 包括内部神经元脉冲发放与外部刺激脉冲输入等. 在事件驱动下, 相关神经元会迅速并行完成对脉冲的计算与传输, 而当没有脉冲发生时, 神经元与神经网络会逐步进入并保持休眠状态. 这与卷积神经网络 (CNN)、循环神经网络 (RNN) 等人工神经网络使用连续且具体的数值进行信息传递是不同的. 因此, 在事件驱动下的脉冲神经网络不仅能够模仿生物神经系统的信息编码与处理过程, 具有稀疏但强大的并行计算能力, 而且具有低功耗的优势. 脉冲神经元将输入累积到膜电压, 当达到具体阈值时进行脉冲发射, 能够进行事件驱动式计算. 然而, 因为 SNN 的训练与推理过程需要模拟微分方程, 因此在实现上还存在较大的困难, 目前还不是较为实用的工具. 目前, SNN 在实时图像处理和音频处理领域得到了一些实际应用. IBM 的 TrueNorth 旨在是通过使用特定硬件模拟神经元, 该硬件可以利用神经元脉冲行为的离散和稀疏特性优势来模拟神经元 [60]. Intel 的 Loihi 芯片在硬件层面上复制神经元组织、通信与学习方式 [61], 展示了在极小样本训练的条件下 Loihi 对危险化学品气味的高识别率, 并推出了基于 Loihi 芯片的神经拟态计算系统 Pohoiki Springs, 包含 1 亿个神经元计算能力. 灵汐科技另辟蹊径, 将较为成熟的 CNN 与 SNN 在可重构计算平台上融为一体, 实现了实时视觉目标探测、目标追踪、自动避障、语音理解控制, 以及自主决策等功能 [62].

2.3.5　小结

指令流驱动经过多年的发展, 尤其是在通用处理器、高级语言, 以及编译模型的支持下, 是目前计算系统主要的驱动方式, 具有高度灵活的特点, 但是指令流的时域串行执行方式导致其性能尤其是效能的进一步提升存在一定困难; 数据流驱动具有先天性并行化特征, 具有高效率低功耗的优势, 但是目前技术还不够成熟, 产业化应用存在一定的困难, 不过与指令流驱动方式的结合为计算系统的进步带来了明显的收益; 配置流与数据流共同驱动是在指令流驱动方式与数据流驱动方式之间的折中: 既具有一定的灵活性, 又保留了较大的性能与效能优势, 尤其是与领域专用计算方式相结合时, 是未来极具

潜力的计算驱动方式; 事件驱动是一种特殊的计算驱动方式, 在脉冲神经网络越来越受到关注的情况下, 是人工智能领域, 尤其是类脑计算方向的重要技术方法. 由上述可以看出, 不同的计算驱动方式在根本上对计算性能、效能, 以及灵活性等方面的影响具有巨大的差异性, 而面向未来应用的计算需求, 可以推测合理的计算驱动方式应当是将其有机的结合起来, 各取其长且避其短, 才能全面提升计算系统的技术指标.

2.4 从计算核心构成形式看体系架构

2.4.1 单核结构

单核结构并不是在单核处理器发明时就有的名称, 而是在双核与多核结构出现后相应派生出来的概念. 顾名思义, 所谓单核结构就是处理器仅包含一个逻辑核心. 在 1971 年英特尔推出第一款微处理器之后, 在随后的 30 年中人们一直致力于单核结构处理器计算能力的发展, 其推动力主要来自于制程工艺的快速进步: 晶体管尺寸不断缩小、集成密度不断升高带来了时钟频率的上升、芯片功能的丰富, 以及制造成本的下降. 但是在 2000 年前后, 研究人员预测, 如果仅依赖制程工艺的发展提升处理器性能, 到 2010 年前后芯片功耗密度将达到火箭发动机的水平, 这不仅带来了巨大的散热压力, 而且芯片本身也是无法承受的. 因此, 在 2005 年以后, 人们不再追求时钟频率增长以及由此带来的性能收益, 转而开始研究多核结构.

2.4.2 同构多核结构

同构多核是指在单个处理器中集成两个或多个同构计算逻辑核心或单个系统中集成两个或多个同构处理器的结构. 虽然根据摩尔定律预测芯片晶体管集成度快速提高, 而集成度的提高意味着时钟频率的提升, 并带来计算性能的提高. 但当时钟频率接近 4 GHz 时人们发现, 单纯通过时钟频率的提升来提高计算性能会遇到无法克服的瓶颈, 例如虽然奔腾 IV 处理器的时钟频率为 3.6 GHz, 但是其计算性能尚不如时钟频率为 3.4 GHz 的产品. 究其原因, 指令流水线太长会造成单位频率对逻辑核心的驱动效率会下降. 单核结构处理器的另一个突出问题是因为时钟频率提升、增加缓存寄存器等原因, 其功耗会大幅上涨, 散热问题以及由此带来的可靠性下降问题也成为制约单核处理器发展的因素.

事实上, 早在 20 世纪 90 年代人们就预测到了这样的问题, 一方面通过降低处理器电压缓解功耗问题, 另一方面则不再追求单纯通过提升时钟频率来提升单核结构处理器的性能, 而是将目光转到了多核心处理器的研究上. 1996 年世界首款多核心处理器原型系统 Hydra [63] 在斯坦福大学 (Stanford University) 诞生. 在随后 10 年中, 双核处理器快速发展, 不仅带来了计算性能的大幅提升, 而且为多任务的并行计算奠定了更好的硬件基础. 伴随着双核以及多核处理器的快速发展, 人们不得不面临另一个难题, 即如何充分利用双核甚至多核处理器的结构优势, 发挥更大的并行计算效力. 这不仅涉及到硬件逻辑结构层面的优化设计, 更重要的是需要软件行业, 尤其是编译器与操作系统的巨大创新. 其中, 良好的可编程性是多核同构处理器在设计与应用层面面临的最大问题, 至今尚无根本的解决方案, 编写并行处理程序并进行调试与优化的能力还非常欠缺.

2.4.3 异构多核结构

异构多核结构是相对于早期以同构多核处理器为核心的通用计算系统而言的. 相对于异构多核结构, 人们提及更多的则是基于异构多核结构的异构计算技术, 而该技术在 20 世纪 80 年代中期就走入了人们的视野 [64]. 虽然随着制程工艺的进步, 通用计算芯片性能快速提升, 但随着信息化革命日趋深入, 数据量爆炸性增长, 应用需求多样化发展, 尤其是近年来大数据、云计算、人工智能等对计算性能

与计算效能要求极高的应用领域快速发展, 基于传统同构多核处理器的通用计算系统已经远远无法满足应用需求. 因此, 人们再次将目光投向同时兼具不同计算优势的异构多核结构及异构计算技术上.

异构多核结构是指将多种具有不同体系架构、不同颗粒度、使用不同类型指令集、不同计算特点的计算单元有机组合在一起所形成的的混合计算结构. 异构计算技术则是在异构多核结构计算系统的基础上, 根据各个计算单元的计算特点或优势对应用任务进行合理划分与映射, 以追求异构多核计算系统技术指标最优化的并行和分布式计算模式. 异构计算的研究主要针对两大类问题: 一方面是异构多核结构计算系统如何构建, 不仅要具有完备的, 能够适应应用计算特点的异构计算核心, 而且需要将各类计算核心有机统一起来以形成计算平台; 另一方面则是计算平台异构资源计算优势的高效利用问题, 包括任务本身的并行性挖掘、异构计算核心计算优势挖掘、基于任务并行性与资源异构性的任务资源管理调度, 以及任务资源匹配映射等. 因此, 异构计算技术是一种包含特殊化计算平台建设与特殊化平台使用的软硬件协同计算技术, 能够将应用任务的并行化模型与计算平台的异构化优势有机结合起来, 以实现最优化计算目标, 不仅是并行化计算与分布式计算技术方向的热点, 而且已经在当前人工智能 [65]、云计算 [66], 以及大数据处理 [67] 等领域获得了广泛的应用.

从异构计算构成方式或异构单元颗粒度角度来说, 异构计算可分为 3 类: (1) 板级集成异构计算, 即不同功能或不同组成的板卡组成异构系统, 通过高带宽连接起来联合解决相关计算问题, 如将 CPU 板卡、DSP 板卡或 FPGA 板卡组合起来解决雷达信号处理问题 [68]; (2) 芯片级 (SoC) 异构计算, 即将不同制程、不同架构的芯片组合在一起, 并联合解决计算问题, 如针对数据压缩问题, 在 IBM POWER9 和 z15 芯片上集成了一块专门用于数据压缩的加速器 NXU, 能够将数据压缩速度, 进而提高芯片处理速度, 并且只占用很小的芯片面积 [69], Centaur Technology 公司高性能深度学习协处理器 NCore 集成在 X86 SoC 上, 是服务器集 CPU 的协处理器, 支持多种数据类型性, 可灵活扩展, 在多个数据集上具有高吞吐量和低延迟的效果 [70]; (3) 超异构计算, 即将很多现有的、在不同节点上已经充分验证的 Chiplet 通过 EMIB, Foveros 这些 2D, 3D 封装技术集成在一个封装模块里. 相比之下, 板级异构计算的优势是相对更加灵活, 但主板与主板之间连接起来体积比较大, 而且连接带宽与功耗都很难达到最优解; 芯片级异构在功耗和性能方面有着一定的优势, 但灵活性上明显不足, 而且要求设计人员对应用负载有非常深的理解, 同时芯片级异构一旦完成就无法更改, 如应用需求发生变化, 则人力、时间成本消耗都非常高. 因此, 超异构计算是在当前制程工艺、设计方法, 以及封装技术研究成果的基础上, 基于异构计算技术思想, 面向未来应用需求而建立的新型异构计算技术, 具有更为广阔的发展前景.

2.4.4 小结

对于传统的处理器, 计算性能等技术指标的提升的途径主要是有两个: 一是提升工作时钟频率, 但随着摩尔定律逐渐失效, 这种方法已经难以为继, 甚至出现了不进反退的困局; 二是扩展同构逻辑核心, 但同样遇到了能耗与散热问题. 对此, 人们一方面采用借鉴数据流驱动计算思想的流处理模式构建了 GPU 等同构众核处理器, 虽然工作时钟频率较低, 但是具有更多的内核以及并行化计算模式; 另一方面则博采各类异构处理器之长, 将合适的应用放在合适的处理器上实现, 通过释放各类处理器最大潜力以求获得系统最优性能.

2.5 从计算逻辑使用方法看体系架构

2.5.1 计算逻辑固定不变

如上述, 不论是通用计算体系架构还是专用集成电路, 采用的是指令流驱动计算方式还是数据流驱动计算方式, 都有一个共同的特点: 即底层的硬件逻辑架构都是固定不变的. 在计算过程中, 要么通

过编译系统与操作系统将根据应用算法的计算流程分解为相应的指令并在时域按顺序下发至底层硬件逻辑完成计算, 要么通过将数据灌入能完成特定计算的固定逻辑架构里并在空域按顺序依次完成计算. 在硬件结构固定的条件下, 如果配合指令流驱动计算方式, 能够实现一切可计算问题, 具有巨大的灵活性收益 (如 CPU 与 GPU 等). 不过因为逻辑结构不可改变, 只能以应用适应结构, 根据不同的应用做出一定程度的计算过程优化, 而且指令流驱动计算需要经历类似取指、译码、访存、执行, 以及数据回写多个步骤, 这种很高的指令执行密度使得真正计算过程 (怎么做) 所占的比重偏低, 而功能化、数据访问等信息的分析过程或者广义指令解码过程 (做什么) 占据了大量时间, 因此其计算效率与能效比较低. 同样, 在硬件结构固定的条件下, 如果配合数据流驱动计算方式, 可直接根据特定应用任务的计算需求定义做什么、怎么做的优化硬件实现方式, 能实现对特定应用任务的计算加速, 具有巨大的性能与效能收益 (如 AISC 等), 但因逻辑结构一旦设计定型就无法改变, 因此其灵活性极低, 且时间成本与经济成本较高.

在特定的历史时期, 硬件结构固定的计算方式在通用计算或专用计算方面极大地推进了计算技术的发展. 但在当前应用任务多样化、计算性能与效能要求高的条件下, 灵活性、计算性能与计算效能均已成为计算系统最主要的评价标准. 因此硬件结构固定的硬件使用方法已经无法满足未来计算技术发展的需求.

2.5.2 计算逻辑可重构

从计算系统发展的历史经验可以看出, 计算性能、计算效能, 以及计算灵活性之间存在着天然的矛盾. 因此, 人们一直在思考是否存在一种结构 X, 既可以像通用处理器一样实现各类应用算法的计算, 或者说在一定范围的应用集合中能灵活切换减少反复设计底层硬件逻辑的次数, 保持一定程度的灵活性, 又可以继承专用集成电路展开的数据流驱动计算方式, 实现批量操作到电路的转换 (包括空间和时间的映射), 从而实现计算资源的高利用率, 获得高性能、高效能及其他原本通过定制电路所获得的收益. 该结构是否存在, 理论研究与应用产品均已给出了相应的答案, 这就是近年来快速发展的可重构计算技术.

可重构计算是指能够实现算法到计算引擎的空间映射, 并在被制造成集成电路后还具备定制能力的计算组织形式 [71]. 与专用集成电路相比, 可重构计算具有更高的灵活性, 在硅后仍然具有计算结构与计算功能定制能力, 与指令流驱动计算相比, 可重构计算具有更高能量效率, 能够实现算法到空域计算结构的映射 [72]. 可重构计算技术的理念最早可以追溯到 20 世纪 60 年代, 加州大学洛杉矶分校 (University of California, Los Angeles) 的 Estrin 就提出 [73]: 计算机可以由负责系统控制的主处理器和一组结构可以重构的逻辑器件组成, 其中可重构逻辑器件根据应用任务的计算流程与计算特点进行裁剪或重构, 以合适的计算结构实现对应用任务的计算加速, 而主处理器则对可重构逻辑器件的重构与计算过程进行控制. 然而, 一方面当时现代集成电路刚刚发明, 制程工艺水平有限, 另一方面基于基于冯 · 诺依曼架构的通用处理器仍然是人们关注的重点, 所以该理念并未受到太多关注.

从 20 世纪 70 年代末开始, 在不到 10 年的时间里, 多家公司相继推出可重构计算器件, 包括 AMD 的 PAL, Lattice 的 GAL, 以及 Xilinx 的 FPGA 等. 这些可编程逻辑器件是早期可重构计算主要形式. 与 PAL, GAL 等器件相比, FPGA 结构有着明显的不同: 基本逻辑单元模块由查找表由查找表与 D 触发器等组成, 基本逻辑单元之间通过金属线互连, 能够实现组合逻辑功能与时序逻辑功能, 并通过向内部静态存储单元加载编程数据实现对逻辑单元功能及逻辑单元之间互连方式的更改, 以实现不同的功能. FPGA 具有逻辑规模大、晶体管使用效率高、计算速度块、编程灵活且可多次重复编程等优势. 虽然目前 FPGA 已经成为极为重要的可重构计算形式, 但是在其发明之初, 通常作为计算系统中的从

属性计算部件, 或单纯作为算法功能验证器件 [74]. 直到 20 世纪 90 年代, 人们逐渐认识到了 FPGA 作为细粒度通用可重构器件的高灵活性与作为基于逻辑门直接定义功能的高效性, 开始以 FPGA 为主要计算部件设计面向应用的计算设备.

然而, 随着 FPGA 的应用日益广泛, 其主要缺点也逐渐显现出来: 其通用可重构特性引入了大量冗余布线资源, 致使工作时钟频率受限, 且资源浪费严重; 其细粒度可重构特性需要编程数据量庞大、开发过程与重构过程时间长、编程难度大、时序规划难等; 其直接基于逻辑门的计算结构构建方式不适合条件操作, 也无法处理多事件等. 受限于 FPGA 等细粒度可重构计算技术的缺点, 人们开始探索新的可重构计算技术途径, 粗粒度可重构计算架构 (CGRA) 逐步发展起来.

CGRA 最早出现于 20 世纪 90 年代 [75], 并在近年来迅速发展. CGRA 之所以能够持续吸引工业界和学术界关注, 是因为它具有接近 ASIC 的能效和性能, 以及制造后的软件可编程性 [76]. 随着 CGRA 技术的流行, 产生了一系列 CGRA 产品, 如 PACT-XPP, PADDI, PipeRench, KressArray, Morphosys, Matrix, REMARC, REMUS 等. CGRA 的体系架构如图 7 所示, 其主要特征包括:

(1) 特定应用领域灵活性. CGRA 具有较强的硅后灵活性, 它的硬件可在运行时由软件定义. 这对于一个甚至多个特定领域来说已经足够灵活, 能满足绝大多数应用的需求. 与通用灵活性的不同之处在于, 特定领域灵活性使硬件满足目标应用需求并将冗余资源最小化, 有效提高计算性能与效能. 因此, 特定应用领域灵活性被认为是 CGRA 在能效和灵活性之间取得平衡的关键原因所在.

(2) 结合空域和时域的计算. 空域上, CGRA 利用并行计算资源和数据传输通道来执行计算. 时域上, CGRA 利用时分复用资源来执行计算. 时空计算的结合为应用提供了一个更加灵活和有力的实现架构. 相对于仅实现时域计算的架构, CGRA 可以避免昂贵的深度流水线和集中式通信开销; 相对于仅实现空域计算的架构, CGRA 可提高面积效率. 因此, 结合空域和时域计算是 CGRA 在不降低灵活性的前提下实现高能量效率和高面积效率的关键原因之一.

(3) 配置和数据同时驱动执行. CGRA 的操作主要由配置流或数据流来驱动, 其中配置定义 PE 操作及其互连. 虽然配置也主要由控制流驱动, 但在每个配置中的操作是并行或是流水的, 发掘了计算并行性, 并且配置驱动的 CGRA 可以通过互连有效地利用显式数据流, 这也是 CGRA 能够实现高灵活、高性能与高能效三者有机平衡的关键原因之一.

以不同的标准对 CGRA 进行分类的方法很多: (1) 以编程模型为标准, CGRA 可分为命令式编程模型和命令式语言编程, 包括顺序化语句、命令或指令序列以及 C 或 C++ 等高级语言、并行编程模型, 包括声明性编程模型、并行/并发编程模型等、透明编程, 即不进行任何静态编译, 采用动态编译技术, 依赖硬件在程序运行时实时翻译或优化常见的程序表示; (2) 以计算模型为标准, CGRA 可分为单配置单数据 (SCSD) 模型、单配置多数 (SCMD) 模型、多配置多数据 (MCMD) 等; (3) 以执行模型为标准 CGRA 可分为静态调度顺序执行 (SSE)、动态调度静态数据流执行 (DSD)、静态调度静态数据流执行 (SSD), 以及动态调度动态数据流执行 (DDD).

目前, CGRA 因复杂的底层计算架构、主流顺序风格软件编程模型与大量并行性 CGRA 架构之间的冲突、面积与功耗以及并行性之间的矛盾、CGRA 底层结构多样化的设计与开发环境, 以及存储墙等原因而面临着架构编程、有限的并行计算、虚拟化, 以及内存效率等方面的技术挑战, 尚待更深入的研究.

邬江兴院士于 2009 年提出并开始研究的拟态计算的概念 [13], 本质上是一种结合异构计算技术优势的系统级可重构计算技术, 其核心理念可以概括为研究和建立最合适的计算模型、使用和构建最合适的处理部件、设计和匹配最合适的体系结构, 从而追求和逼近最理想的综合效能, 已经在多个领域广泛应用 [14~19].

2.5.3 小结

计算逻辑固定不变的体系架构主要包括两类: 一是配之以相应的指令集与自动化编译工具, 以实现计算任务到计算逻辑的映射, 虽然具有非常高的灵活性, 但是在空间上其所必须具备的取指、译码等非算术逻辑单元会占用大量资源, 而且在时间上其计算流程中计算指令执行本身所占用的时间比例很小, 造成了其能量效率的低下; 二是直接将特定的应用计算结构映射为固化的电路, 并配之以数据流驱动计算方式, 虽然具有非常高的能量效率, 但是其功能无法改变, 具有较高的经济成本与时间成本. 因此, 计算逻辑固定不变无法解决灵活性与能量效率之间的矛盾. 对此, 计算逻辑可重构, 包括细粒度重构与粗粒度重构、静态重构与动态重构等不同方式已经成为未来计算体系架构发展的重要方向.

3 未来计算系统的能力需求

近年来, 信息技术特别是人工智能技术的飞速发展与应用, 对现代战争、工业范式, 以及人们的日常生活产生了深刻的影响. 随着信息革命时代的到来, 数据量呈爆炸式增长趋势, 针对不同应用任务, 各类算法层出不穷, 嵌入式设备、边缘终端与移动终端广泛应用, 对计算系统提出了更高的要求.

从任务需求角度看, 虽然人工智能应用在未来计算系统中占据的比重越来越大, 但在当前及未来很长时间内, 传统科学计算仍然是计算系统任务的重要组成部分: 一方面未来计算系统不仅需要具备支撑诸如信号/信息处理、网络数据包处理等传统科学计算的能力, 而且必须能够承担文本处理、语音分析, 以及图像识别等人工智能处理任务; 另一方面从原始数据到人工智能的输入之间需要大量的基于科学计算的预处理操作, 如原始图像的滤波与增强、原始语音的预加重与加窗等. 因此, 未来计算系统中科学计算与人工智能密不可分, 不仅有合二为一的应用需求, 而且科学计算需要借助人工智能方法向智能化方向发展, 而目前主流人工智能算法的发展又与科学计算的进步强相关, 两者具有强烈的相互支撑、融合发展的必要性.

从计算需求角度来看, 不论是科学计算技术还是人工智能技术, 待处理数据量均呈大幅增长趋势, 从数据中快速挖掘出感兴趣的信息难度越来越大, 因此, 海量数据的实时处理是计算系统面临的第 1 个挑战. 数据获取方式与数据形态日趋多样化, 如语音、文本、图像等, 对不同类型数据信息处理宜采用不同的计算方法, 因此, 计算方法能够灵活调整, 实现多功能一体化是计算系统面临的第 2 个挑战. 嵌入式设备与便携式移动终端的广泛应用, 对计算系统的功耗提出了苛刻的要求, 因此, 在高性能前提下大幅降低功耗是计算系统面临的第 3 个挑战. 在人工智能繁荣发展的时代, 不论是人工智能本身还是传统的科学计算模式, 其计算过程与资源分配均需要自适应优化调整, 向智能化方向发展, 因此, 实现计算的智能化管理是计算系统面临的第 4 个挑战. 因此, 从计算需求角度, 计算系统必须具备大数据量实时处理、计算结构灵活可变、计算效能要求苛刻, 以及计算过程智能优化等能力, 简而言之, 必须同时满足高灵活、高性能、高可靠, 以及高效能等四位一体的 “多、快、好、省” 的信息处理需求, 计算性能、效能、灵活性, 以及智能化水平已经成为衡量未来计算系统的主要技术指标.

从系统应用角度看, 随着信息化革命的不断深入, 特别是 5G、物联网, 以及云计算等技术的快速发展与应用, 不论是军事作战体系还是民用信息系统, 均呈现出由分离向融合、由节点向网络、由单装到集群、由有控向自主、由单域向多域发展的趋势. 因此, 未来计算系统在应用过程中, 除了计算节点必须能够以 “多、快、好、省” 的计算能力同时应对科学计算与人工智能两大类应用外, 一方面应当具有开放性体系架构, 实现对新器件、新板卡乃至新计算系统的快速兼容能力, 解决当前信息系统 “牵一发而动全身” 的迭代升级困难问题; 另一方面, 应当具备随时随地按需组网应用的能力, 每个系统既可以单独运行, 也可按照应用需求作为计算节点承担相应的计算任务.

4 软件定义计算体系架构

4.1 软件定义计算体系架构的提出

在过去的几十年中, 在应用任务变化、制程工艺进步, 以及设计思想演进等各方面合力的推动下, 计算系统设计与应用一直在发现问题、解决问题的循环过程中快速发展. 其中, 作为计算系统设计的基因性技术, 计算体系架构的设计一直是一条极其重要的技术途径. 但是如上述, 从不同的角度来看, 每一种体系结构在不同的发展阶段既有其存在的合理性, 也有其不可避免的缺点与适用范围. 通用计算体系架构因采用底层硬件逻辑固定不变, 指令流驱动的时域计算方式, 因此其灵活有余, 但性能提升已遇到瓶颈, 尤其功耗墙更是难以逾越. 专用集成电路体系架构采用面向特定算法的底层硬件逻辑优化设计与数据流驱动的空域计算方式, 因此其性能与效能非常可观, 但灵活性极差, 也带来了较高的人力与经济成本. 异构计算从板卡级异构、芯片级异构再到内核级超异构, 为计算技术的发展带来了新的发展方向, 但是其异构颗粒度均相对较大, 针对不同的计算需求, 计算结构的优化还不够彻底, 性能与效能的提升幅度受限. 而可重构计算不论是细粒度的 FPGA 与粗粒度的 CGRA, 都采用硬件逻辑结构重构, 配置流与数据流共同驱动的方式实现计算过程, 对计算密集型应用具有较高的加速比, 但是对计算简单、控制复杂的应用场景适应性较差. 综上, 一方面基于不同体系架构设计的计算器件或计算系统虽有其不可避免的基因缺陷, 但也有其擅长的计算类型; 另一方面相比于逻辑结构固定不变的计算架构来说, 以动态灵活变结构计算方式实现对不同应用任务的计算结构的匹配能够取得更高的能效比. 因此, 如果能够扬长避短, 将各类计算架构优势集合起来, 面向领域内计算流程与计算模块具有巨大相似性的应用任务集合, 基于异构的、具有混合颗粒度的结构化逻辑模块, 以软件灵活定义方式建立一种系统级可重构的计算架构, 理论上能够实现计算系统性能、效能、灵活性甚至智能化方面的有机平衡与综合提升.

4.2 软件定义计算体系架构的内涵

从计算体系架构发展历程梳理以及现有体系架构问题分析的基础上, 不论是学术界还是工业界已经普遍认识到体系架构的创新是实现计算系统进一步发展的重要途径. 而从近年来国内外研究人员的工作成果来看, 面向特定应用领域的软件定义计算体系架构已经成为未来计算体系架构演进的重要方向.

软件定义计算是指面向特定领域, 以软件定义互连为中心, 在硬件资源数字化、标准化与虚拟化的基础上, 将计算能力以异构的混合颗粒度可重构池化节点的形式提供给用户, 并根据多样化的用户应用需求, 以软件灵活可定义与主动感知可重构相结合的控制方式, 通过对硬件资源的高效编排、动态优化, 以及综合利用, 在实现灵活多样的定制化功能的前提下, 实现系统运行效率和能量效率最大化的计算方式.

软件定义的特点是硬件资源虚拟化、互连结构柔性化、系统软件平台化, 以及应用软件多样化. 其中硬件资源虚拟化是指将各种实体硬件资源, 包括通用处理部件、专用集成电路、混合颗粒度可重构构件化计算资源, 以及分布式多层级存储资源等进行抽象化, 打破其物理形态的不可分割性, 以便根据应用计算需求通过灵活重组、重用发挥其最大效能. 互连结构柔性化是指在子系统级、板卡级、芯片级, 以及片内混合颗粒度构件级等多个层级实现软件可定义的互连结构, 包括互连协议可定义、互连协议可转换、互联拓扑可转换、互连带宽可调整、互联端口可定义、互连内容可处理, 以及交换模式可定义等, 达到各个层级资源之间数据传输的灵活性与可靠性. 系统软件平台化是指通过基础软件对硬件资源进行统一管控、按需分配按需配置与分配, 并通过标准化的编程接口解除上层应用软件和

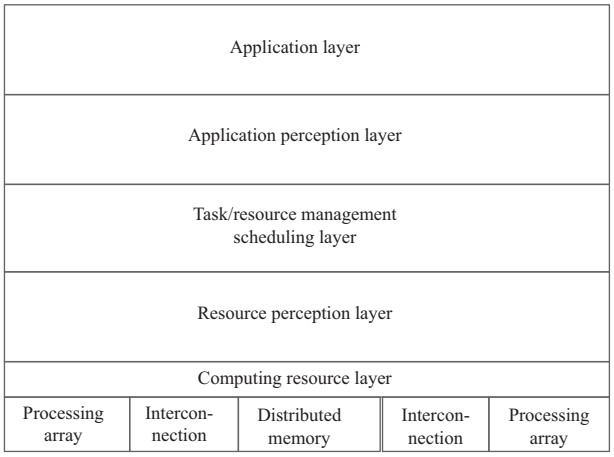

图 8 软件定义计算体系架构层次化

Figure 8 Software defined computing architecture is hierarchical

底层硬件资源之间的紧耦合关系, 使其可以各自独立演化. 应用软件多样化是指在成熟的平台化系统软件解决方案的基础上, 应用软件可不受硬件资源约束, 实现更多的功能, 对外提供更为灵活高效的和多样化的服务.

软件定义计算体系架构层次化表示如图 8 所示. 结构共分为 5 层, 第 1 层为业务层, 包括多样化的应用集合; 底层为计算资源, 包括具有混合颗粒度的异构构件化处理阵列、通用处理部件、专用集成电路等计算资源、分布式层次化存储等存储资源, 以及基于软件定义互连的柔性互连结构等互连资源; 第 2 和 5 层为业务感知层和资源感知层, 它提供资源、应用业务的属性、状态等信息; 第 3 层为任务/资源管理调度层, 依据感知信息、知识库, 通过软件定义与主动认知相结合的方式动态进行决策, 提供应用所需的高效结构. 该计算体系的特点包括: (1) 减少了虚拟化层数, 实现了软件定义更加直接地面向晶体管资源, 能够有效提高晶体管执行效率; (2) 底层计算资源采用异构化方式设计, 既指处理单元 (PE) 实现所依托的计算资源类型不同, 也指 PE 本身的实现方式异构; (3) 采用分布式层次化存储结构, 实现了近存储计算, 具有更高的数据存取效率; (4) 体系架构采用的软件定义互连结构, 改变了传统总线式互连结构, 具有更高的灵活性.

软件定义计算的技术要点包括旨在实现数字化与标准化计算资源池的混合颗粒度计算构件设计、实现与布局方法; 旨在实现数据灵活存储的分布式多层级的软件定义存储结构; 旨在实现池化计算与存储资源高效灵活连接的软件定义互连结构; 旨在实现计算资源、存储资源, 以及互连资源融合一体的硬件体系架构; 旨在实现资源有效管理的池化资源抽象与虚拟化技术; 旨在实现应用任务向资源映射的并行编译技术; 旨在实现计算结构实时优化的流程控制与动态调度技术以及为并行编译与动态优化提供支撑的环境感知与智能决策技术等. 这些技术之间并非孤立, 而是相互关联的整体, 如图 9 所示.

5 总结

本文首先从应用适应性、系统重心变化、计算驱动方式、计算核心构成形式, 以及计算逻辑使用方法 5 个方面对体系架构发展历程进行了梳理, 并总结了从不同角度看体系架构时存在优缺点, 然后从任务需求、计算需求, 以及系统应用需求等方面对未来计算系统应具备能力特征进行了分析, 最后

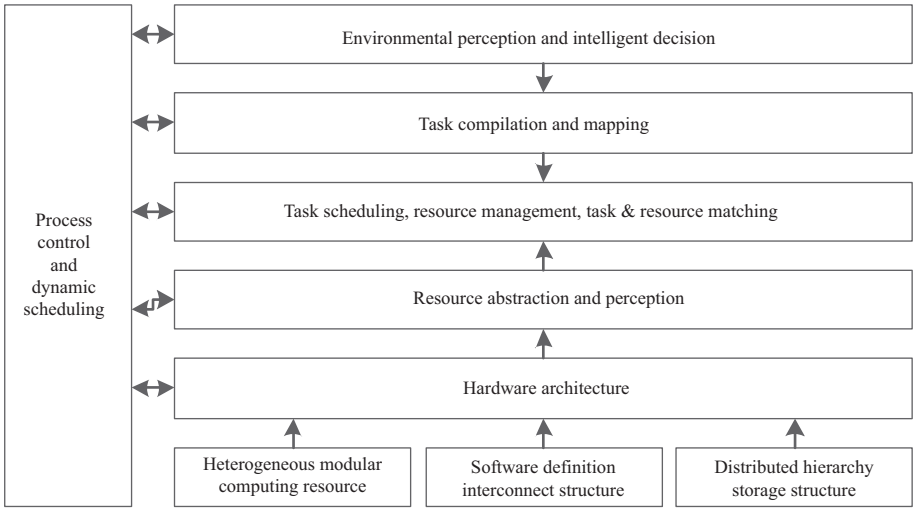

图 9　软件定义计算技术要点关系图

Figure 9　Relationship diagram of software-defined computing technology key points

从当前各类体系架构的缺陷出发, 指出了将各类体系架构优势综合起来的软件定义计算体系架构的基本内涵及其相关的研究内容与关键技术点, 为后续软件定义计算体系架构的研究与发展提出了一条可行的技术途径.

参考文献

1 Bohr M. A 30 year retrospective on Dennard's MOSFET scaling paper. IEEE Solid-State Circ Newsl, 2007, 12: 11–13

2 Amdahl G M. The structure of SYSTEM/360, part III: processing unit design considerations. IBM Syst J, 1964, 3: 144–164

3 Lv P, Liu Q R, Wu J X, et al. New generation software-defined architecture. Sci Sin Inform, 2018, 48: 315–328 [吕平, 刘勤让, 邬江兴, 等. 新一代软件定义体系架构. 中国科学: 信息科学, 2018, 48: 315–328]

4 Zhu M. Research on the application adaptability evaluation methods of internet architecture. Dissertation for Ph.D. Degree. Beijing: Tsinghua University, 2014 [朱敏. 互联网体系结构应用适应性评估方法研究. 博士学位论文. 北京: 清华大学, 2014]

5 Wei S J, Liu L B, Yin S Y. Reconfigurable Computing. Beijing: Science Press, 2014 [魏少军, 刘雷波, 尹首一. 可重构计算. 北京: 科学出版社, 2014]

6 He K C, Shi H B. Harvard structure-based single cycle stack processor design. Microelectron Comput, 2013, 30: 66–70 [何凯成, 施慧彬. 基于哈佛结构的单周期堆栈处理器设计. 微电子学与计算机, 2013, 30: 66–70]

7 Chen X. Eight-stage pipeline structure design in a CPU. Dissertation for Master's Degree. Xiangtan: Xiangtan University, 2018 [陈宪. 一种 CPU 中八级流水线结构设计. 硕士学位论文. 湘潭: 湘潭大学, 2018]

8 Wang H. Study on multi-thread parallel programming method based on multi-core environment. Dissertation for Master's Degree. Zhengzhou: Zhongyuan University of Technology, 2014 [王晗. 基于多核环境下的多线程并行程序设计方法研究. 硕士学位论文. 郑州: 中原工学院, 2014]

9 Wang Q Y, Yu N M, Lu W. Reseach of scalable 32 bit microprocessor for multiple instruction launch technology. Comput Eng Appl, 2011, 47: 69–73 [王琦瑛, 余宁梅, 路伟. 可扩展 32 位微处理器指令多发射技术研究. 计算机工程与应用, 2011, 47: 69–73]

10 Tan H J. Changes and development of computer instruction systems. Sci Technol Inform (Acad Res), 2007, 15: 172–173 [谈怀江. 计算机指令系统的变化及发展. 科技信息 (学术研究), 2007, 15: 172–173]

11 An B L. Analysis of the development and prospect of AI chips. Micro/nono Electron Intell Manu, 2020, 2: 91–94 [安宝磊. AI 芯片发展现状及前景分析. 微纳电子与智能制造, 2020, 2: 91–94]

12 Sha Z Y. ASIC for intelligent instrument and its application. Integr Circ Appl, 1996, 4: 6–12 [沙占友. 智能仪表专用集成电路及其应用. 集成电路应用, 1996, 4: 6–12]

13 Wu J X. Meaning and vision of mimic computing and mimic security defense. Telecommun Sci, 2014, 30: 2–7 [邬江兴. 拟态计算与拟态安全防御的原意和愿景. 电信科学, 2014, 30: 2–7]

14 Xi S X, Zhang W N, Zhou Q L, et al. High throughput implementation of SHA-512 on mimic computers. Comput Eng Sci, 2018, 40: 1344–1350 [席胜鑫, 张文宁, 周清雷, 等. 基于拟态计算机的 SHA512 算法高吞吐量实现, 计算机工程与科学, 2018, 40: 1344–1350]

15 Zhang W W, Deng J W, Xu W. a pattern recognition system and method based on mimic high performance computing. CN107908473A, 2018-04-13 [张文文, 邓佳伟, 徐稳. 一种基于拟态高性能计算的模式识别系统及方法. CN107908473A, 2018-04-13]

16 Tan J, Zhou Q L, Si X M, et al. Implementation and improvement of full-pipeline MD5 algorithm based on mimic computer. J Chinese Comput Syst, 2017, 38: 1216–1220 [谭健, 周清雷, 斯雪明, 等. 全流水架构 MD5 算法在拟态计算机上的实现及改进. 小型微型计算机系统, 2017, 38: 1216–1220]

17 Liang H, Chen F C, Ji X X, et al. Development status and applied research on mimic technologies for space-ground integration information network. Sci Sin Inform, 2019, 49: 799–818 [梁浩, 陈福才, 季新生, 等. 天地一体化信息网络发展与拟态技术应用构想. 中国科学: 信息科学, 2019, 49: 799–818]

18 Gao Y Z, Wang J M, Lei Z Y, et al. Mimic signal processing method for distributed opportunity array radar. Modern Radar, 2021, 43: 1–6 [高彦钊, 王建明, 雷志勇, 等. 分布式机会阵雷达拟态信号处理方法. 现代雷达, 2021, 43: 1–6]

19 Gao Y Z, Zhang Y L, Shen J L, et al. Software-defined radar signal processing architecture based on mimic computing. Commun Comput Technol, 2020, 40: 10–14 [高彦钊, 张永丽, 沈剑良, 等. 基于拟态计算的软件化雷达信号处理体系架构. 通信与计算技术, 2020, 40: 10–14]

20 Rondeau T. Electronics resurgence initiative: architectures. DARPA Microsystems Technology Office, 2017. https://www.darpa.mil/work-with-us/electronics-resurgence-initiative

21 Shen W. Software defined hardware for data intensive computation. DARPA I2O, 2017. https://www.darpa.mil

22 Hennessy J L, Patterson D A. A new golden age for computer architecture. Commun ACM, 2019, 62: 48–60

23 Yin S Y, Peng O Y, Yang J X, et al. An ultra-high energy-efficient reconfigurable processor for deep neural networks with binary/ternary weights in 28nm CMOS. In: Proceedings of IEEE Symposium on VLSI Circuits, 2018. 37–38

24 Ang J, Kemp K, Robertson D, et al. Decadal plan for semiconductors. U.S. Semiconductor Research Corporation, 2021. https://www.eenewseurope.com/sites/default/files/documents/whitepapers/decadal-plan-full-report.pdf?token=%7Bcontactfield=hashmail%7D

25 Amdahl G M. Validity of single-processor approach to achieving large-scale computing capability. In: Proceedings of AFIPS Spring Joint Computer Conference, 1967. 483–485

26 Wen P. Studies on the PIM architectures and techniques for scientific applications. Dissertation for Ph.D. Degree. Changsha: National University of Defense Technology, 2007 [温璞. 面向科学计算的 PIM 体系结构技术研究. 博士学位论文. 长沙: 国防科技大学, 2007]

27 Liu W. Near-data processing-based compaction optimization in LSM-tree-based key-value stores. Dissertation for Ph.D. Degree. Hefei: Anhui University, 2019 [刘伟. 基于近数据计算的 LSM-tree 键值存储系统 Cpmpaction 优化方法. 博士学位论文. 合肥: 安徽大学, 2019]

28 Mao H Y, Shu J W, Li F, et al. Development of processing-in-memory. Sci Sin Inform, 2021, 51: 173–205 [毛海宇, 舒继武, 李飞, 等. 内存计算研究进展. 中国科学: 信息科学, 2021, 51: 173–205]

29 Li C, Ma S, Wang L, et al. A survey on architecture for three-dimensional networks-on-chip. Chinese J Comput, 2016, 39: 1812–1828 [李晨, 马胜, 王璐, 等. 三维片上网络体系结构研究综述. 计算机学报, 2016, 39: 1812–1828]

30 Fen G, Ning W. Simulation and performance analysis of network on chip architectures. Trans Nanjing Univ Aeronaut Astronaut, 2010, 27: 326–332

31 Matsutani H, Koibuchi M, Yamada Y, et al. Fat H-tree: a cost-efficient tree-based on-chip network. IEEE Trans Parallel Distrib Syst, 2009, 20: 1126–1141

32 Wang W, Qiao L, Tang Z Z. Survey on the networks-on-chip interconnection topologies. Comput Sci, 2011, 38: 1–6 [王炜, 乔林, 汤志忠. 片上网络互连拓扑综述. 计算机科学, 2011, 38: 1–6]

33 Sheng M, Jerger N E, Wang Z Y. Whole packet forwarding: efficient design of fully adaptive routing algorithms for networks-on-chip. In: Proceedings of IEEE International Symposium on High-Performance Comp Architecture, 2012

34 Ma S, Jerger N E, Wang Z. DBAR: an efficient routing algorithm to support multiple concurrent applications in networks-on-chip. SIGARCH Comput Archit News, 2011, 39: 413–424

35 Ma S, Wang Z Y, Liu Z L, et al. Leaving one slot empty: flit bubble flow control for torus cache-coherent NoCs. IEEE Trans Comput, 2015, 64: 763–777

36 Shi H, Huang C B. Flow management based on networks-on-chip power control. Modern Comput, 2019, 18: 18–23 [石辉, 黄朝兵. 基于片上网络功率控制的流量管理. 现代计算机, 2019, 18: 18–23]

37 Wang C H, Sun Q, Zhu X Y, et al. Design of fault-tolerant mechanism based on dimension-ordered routing for NoC. J Hefei Univ Technol (Natl Sci), 2020, 43: 1346–1351 [王春华, 孙琦, 朱新宇, 等. 基于维序路由的片上网络容错机制设计. 合肥工业大学学报 (自然科学版), 2020, 43: 1346–1351]

38 Lin X X, Wang J S, Lin S S, et al. Low overhead fault tolerant transmission mechanism for network-on-chip. Microelectron Technol, 2017, 43: 33–35 [林辛鑫, 王君实, 林水生, 等. 低开销片上网络容错传输机制. 电子技术应用, 2017, 43: 33–35]

39 Pan T. The development and research of computer instruction system. Heilongjiang Sci Technol Inform, 2014, 31: 177 [潘谈. 计算机指令系统的发展与研究. 黑龙江科技信息, 2014, 31: 177]

40 Waterman A, Lee Y, Avizienis R, et al. The Risc-V instruction set. In: Proceedings of Poster at the Symposium on High Performance Chips, 2013

41 Tan Z X, Zhang L, Patterson D, et al. PicoRio: an open-source, RISC-V small-board computer to elevate the RISC-V software ecosystem. Tsinghua Sci Technol, 2021, 26: 384–386

42 Kumar V B Y, Deb S, Gupta N, et al. Towards designing a secure RISC-V system-on-chip: ITUS. J Hardw Syst Secur, 2020, 4: 329–342

43 Miyazaki H, Kanamori T, Islam M A, et al. RVCoreP: an optimized RISC-V soft processor of five-stage pipelining. IEICE Trans Inf Syst, 2020, 103: 2494–2503

44 Dennis J B. Programming generality, parallelism and computer architecture. In: Proceedings of International Federation for Information Processing Congress, 1968. 484–492

45 Dennis J B. First version of a data flow procedure language. In: Proceedings of Symposium on Programming, 1974. 362–376

46 Dou Y, Wang J L, Su Y H, et al. The impact of data flow computing thinking on the development of computer architecture. Sci Sin Inform, 2020, 50: 1697–1713 [窦勇, 王嘉伦, 苏华友, 等. 从计算体系结构发展历程看数据流计算思想. 中国科学: 信息科学, 2020, 50: 1697–1713]

47 Mattson P R. A programming system for the imagine media processor. Dissertation for Ph.D. Degree. Stanford: Stanford University, 2002

48 Khailany B, Dally W J, Kapasi U J, et al. Imagine: media processing with streams. IEEE Micro, 2001, 21: 35–46

49 Taylor M, Psota J, Saraf A, et al. Evaluation of the raw microprocessor: an exposed-wire-delay architecture for ILP and streams. In: Proceedings of International Symposium on Computer Architecture, 2004. 2–13

50 Dally W J, Labonte F, Das A, et al. Merrimac: supercomputing with streams. In: Proceedings of Conference on High Performance Computing (Supercomputing), 2003

51 Suettlerlein J, Zuckerman S, Gao G R, et al. An implementation of the Codelet model. In: Proceedings of International Conference on Parallel Processing, 2013. 633–644

52 Suetterlein J. DARTS: a runtime based on the Codelet execution model. Dissertation for Master's Degree. Newark: University of Delaware, 2014

53 Ye X C, Tan X, Wu M, et al. An efficient dataflow accelerator for scientific applications. Future Generation Comput Syst, 2020, 112: 580–588

54 Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Commun ACM, 2008, 51: 107–113

55 Lu J W, Wu H, Chen H, et al. A performance optimization method based on dynamic topology for Stream computing and its implementation in Storm. ACTA Electron Sin, 2020, 48: 878–890 [陆佳炜, 吴涵, 陈烘, 等. 一种基于动态拓扑的流计算性能优化方法及其在 Storm 中的实现. 电子学报, 2020, 48: 878–890]

56 Zaharia M, Chowdhury M, Franklin M J, et al. Spark: cluster computing with working sets. In: Proceedings of IEEE International Conference on Cloud Computing Technology and Science, 2010

57 Carbone P, Katsifodimos A, Ewen S, et al. Apache flink: stream and batch processing in a single engine. IEEE Data Eng Bull, 2015, 36: 28–33

58 Lu Y, Liu L B, Zhu J F, et al. Architecture, challenges and applications of dynamic reconfigurable computing. J Semicond, 2020, 41: 021401

59 Chin S A, Sakamoto N, Rui A, et al. CGRA-ME: a unified framework for CGRA modelling and exploration. In: Proceedings of the 28th International Conference on Application-Specific Systems, 2017

60 Cassidy A S, Merolla P, Arthur J V, et al. Cognitive computing building block: a versatile and efficient digital neuron model for neurosynaptic cores. In: Proceedings of International Joint Conference on Neural Networks, 2014

61 Davies M, Srinivasa N, Lin T H, et al. Loihi: a neuromorphic manycore processor with on-chip learning. IEEE Micro, 2018, 38: 82–99

62 Pei J, Deng L, Song S, et al. Towards artificial general intelligence with hybrid Tianjic chip architecture. Nature, 2019, 572: 106–111

63 Olukotum K, Nayfeh B A, Hammond L, et al. The case for a single-chip multiprocessor. In: Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems, 1996

64 Lu X D. Dissecting heterogeneous computing. China Comput User, 1999, 12: 29–31 [陆鑫达. 剖析异构计算. 中国计算机用户, 1999, 12: 29–31]

65 Hu S Q. Research and implemetation of image recognition acceleration algorithm based on heterogeneous platform. Dissertation for Master's Degree. Dalian: Dalian University of Technology, 2020 [胡绍齐. 基于异构平台的图像识别加速算法研究与实现. 硕士学位论文. 大连: 大连理工大学, 2020]

66 Fati S M, Jaradat A K, Abunadi I, et al. Modelling virtual machine workload in heterogeneous cloud computing platforms. J Inf Tech Res, 2020, 13: 156–170

67 Huang J B. Design and implementation of a big data intelligent analysis platform for heterogeneous computing. Dissertation for Master's Degree. Beijing: Beijing University of Posts and Telecommunications, 2020 [黄建博. 面向异构计算的大数据智能分析平台的设计与实现. 硕士学位论文. 北京: 北京邮电大学, 2020]

68 Lin D. Software and hardware implementation of signal processing for shipborne fire control radar. Dissertation for Master's Degree. Xi'an: Xidian University, 2020 [林达. 舰载火控雷达信号处理的软硬件实现. 硕士学位论文. 西安: 西安电子科技大学, 2020]

69 Abali B, Blaner B, Reilly J, et al. Data compression accelerator on IBM POWER9 and z15 processors. In: Proceedings of the 47th Annual International Symposium on Computer Architecture, 2020

70 Henry G, Palangpour P, Thomson M, et al. High-performance deep-learning coprocessor integrated into x86 SoC with server-class CPUs. In: Proceedings of the 47th Annual International Symposium on Computer Architecture, 2020. 15–26

71 Li Z Q. Design and optimization of energy-efficient coarse grained reconfigurable architecture for block cipher algorithm. Dissertation for Master's Degree. Nanjing: Southeast University, 2017 [李兆奇. 面向分组密码算法的粗粒度可重构架构高效能设计与优化. 硕士学位论文. 南京: 东南大学, 2017]

72 Dehon A, Wawrzynet J. Reconfigurable computing: what, why, and implications for design automation. In: Proceedings of Design Automation Conference, 1999. 610–615

73 Estrin G. Organization of computer systems: the fixed plus variable structure computer. In: Proceedings of Western Joint IRE-AIEE-ACM Computer Conference, 1960. 33–40

74 Luo S. Architecture research and implementation on reconfigurable compting system. Dissertation for Master's Degree. Hefei: University of Science and Technology of China, 2006 [罗赛. 可重构计算系统体系结构研究与实现. 硕士学位论文. 合肥: 中国科学技术大学, 2006]

75 Hartenstein R W, Hirschbiel A G, Riedmuller M, et al. A novel ASIC design approach based on a new machine paradigm. IEEE J Solid-State Circ, 1991, 26: 975–989

76 Wei S J, Li Z S, Zhu J F, et al. Reconfigurable computing: toward software defined chips. Sci Sin Inform, 2020, 50: 1407–1426 [魏少军, 李兆石, 朱建峰, 等. 可重构计算: 软件可定义的计算引擎. 中国科学: 信息科学, 2020, 50: 1407–1426]

Review and thoughts on the development of computing architecture

Yanzhao GAO*, Jiangxing WU, Qinrang LIU, Jianliang SHEN, Ke SONG & Fan ZHANG

Institute of Information Technology, Information Engineering University, Zhengzhou 450003*, China*
* Corresponding author. E-mail: buaagaoyz@sina.com

Abstract With the end of Moore's law and Dennard's scaling law, it is becoming increasingly difficult to improve the performance and efficiency of computing systems by relying on the progress of the integrated circuit (IC) manufacturing process. The evolution of computing architecture has become an important technical approach for the development of computing systems in the future. In this article, the evolution of the architecture is first reviewed from different perspectives, including the application adaptability, computation-driven mode, changes in the center of the computing system, the composition of the computing cores, and computing logic usage. The advantages and disadvantages of different architectures are summarized, and then the ability to computing systems in the future under the conditions of rapid development of manual intelligence and big data is analyzed. Finally, a software-defined computing architecture is proposed, and its key research contents and key technologies are summarized, which provides a feasible technical approach for the future development of computing architectures.

Keywords architecture, software-defined computing, domain-specific architecture, heterogeneous computing, reconfigurable computing