中国科学：信息科学
SCIENTIA SINICA Informationis
《中国科学》杂志社
SCIENCE CHINA PRESS
评述

从计算机体系结构发展历程看数据流计算思想

窦勇[1]，王嘉伦[2]，苏华友[1*]，徐辰[2]，宫晓利[3]，阳王东[4]，翁楚良[2]，李战怀[5]，李肯立[4]，于戈[6]，周傲英[2]

1. 国防科技大学计算机学院, 长沙 410073
2. 华东师范大学数据科学与工程学院, 上海 200062
3. 南开大学网络空间安全学院, 天津 300071
4. 湖南大学信息科学与工程学院, 长沙 410082
5. 西北工业大学计算机学院, 西安 710072
6. 东北大学计算机科学与工程学院, 沈阳 110169
* 通信作者. E-mail: shyou@nudt.edu.cn

收稿日期: 2020–04–15; 接受日期: 2020–05–25

国家重点研发计划 (批准号: 2018YFB1003400) 资助项目

摘要 在计算机体系结构发展历程中, 冯·诺依曼 (von Neumann) 计算机结构一直是计算机系统的主流架构. 谈及非冯计算机体系结构时, 数据流计算机无疑是被提及最多的. 本文从计算机体系结构发展历程的角度, 分析数据流计算思想在计算机体系结构创新发展过程中发挥的重要作用. 本文首先回顾数据流计算思想、分析早期数据流计算机的局限性; 之后分析在现代中央处理器 (central processing unit, CPU) 技术中所借鉴的数据流计算思想, 乱序执行和多线程技术; 进一步介绍流计算思想、流处理器技术和图形处理器 (graphics processing unit, GPU) 中的数据流计算思想; 然后针对大数据智能化时代计算机系统的发展变化分析数据流计算思想的运用. 最后总结数据流计算思想运用规律, 展望未来发展趋势.

关键词 数据流, 大数据, 异构计算, GPU, 智能计算

1 引言

电子计算机无疑是人类最伟大的发明之一, 代表着人类追求智力自动化的一个新阶段. 它开启了人类社会进入信息化时代的大门, 深刻地影响着人类活动的各个领域, 极大地推动了人类社会的进步. 现代计算机的诞生与发展是现代多门科学与技术进步的产物, 其中半导体晶体管及集成技术是计

引用格式: 窦勇, 王嘉伦, 苏华友, 等. 从计算机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, doi: 10.1360/SSI-2020-0092
Dou Y, Wang J L, Su H Y, et al. Reviewing data flow computing thinking from the development of computer architecture (in Chinese). Sci Sin Inform, 2020, doi: 10.1360/SSI-2020-0092

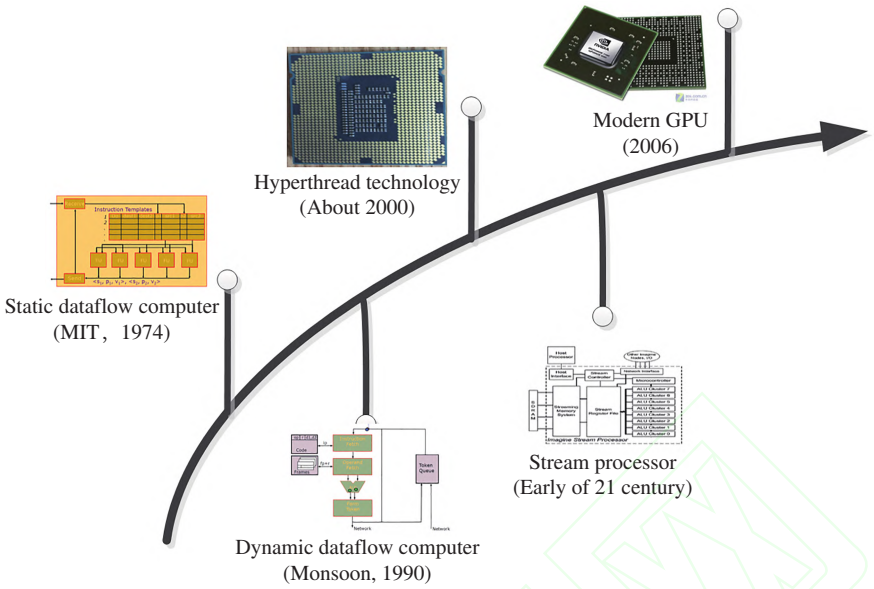

图 1　(网络版彩图) 数据流技术在体系结构中的发展历程

Figure 1　(Color online) The history of data flow computing on architecture

算机发展的物质基础, 布尔代数与图灵机通用计算模型成为计算机系统的理论基础, 冯 · 诺依曼 (von Neumann) 计算机结构为现代计算机设计奠定了技术基础.

自计算机诞生以来, 计算机系统研究人员为提升计算速度, 一直百折不挠、不懈努力. 几十年间学术界、产业界开展了多种多样的计算机体系结构探索, 产生了很多令人耳目一新的成果, 人们熟知的冯 · 诺依曼结构瓶颈被反复诟病, 但是冯 · 诺依曼计算机结构依然是当前计算机系统的主流架构. 在谈及非冯计算机体系结构时, 数据流计算机无疑是被提及最多的. 虽然纯粹的数据流计算机没能成功产业化, 但是数据流计算机的威名像武林传说中的隐形大侠一样, 似乎随时都有可能于水火中拯救苍生. 那么如何评价数据流计算机体系结构呢?

从历史发展的角度来看, 数据流计算思想回答了 “一个运算操作能够被执行的充分条件是什么” 这个科学问题, 即数据就绪开始计算, 同时提出了用数据流图描述计算任务的具体方法. 在当时这些无疑是革命性的创新思想, 其创新之处就在于打破了传统串行执行指令的思维禁锢, 将并行执行的思想带到了计算机设计者面前. 数据流计算机是探索利用数据流计算思想提升计算机系统性能的实践者和先驱者.

从计算机体系结构的角度来看, 虽然纯粹的数据流计算机没有成功, 但是数据流计算思想一直在计算机体系结构创新发展过程中发挥着至关重要的作用, 图 1 简要回顾了数据流技术在计算机体系结构发展中的应用. 计算机体系结构学科包括开展计算机系统设计的科学与工程技术, 其中计算机系统的平衡设计是其核心理念. 平衡设计思想可以体现在用增加硬件成本, 提升系统性能; 也可能体现在为了控制功耗, 损失一定的性能; 也包括为了降低编译难度, 约束用户编程语言; 还可能体现在为了增加容错能力, 在空间维度硬件冗余和时间维度重复执行之间进行权衡. 本文从计算机体系结构平衡设计视角考察数据流计算思想, 我们看到在每个历史时期数据流计算思想都以多种形式在计算机系统性能提升中发挥着重要作用. 如果说控制流计算思想是革命性的, 它推动人类从机械化计算器时代走向了通用计算机时代; 那么数据流计算思想的影响是渗透性的, 在提高计算机系统计算效率的历史长河

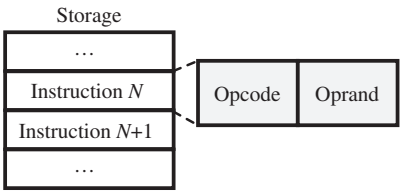

图 2 控制流指令序列和指令

Figure 2 The instruction sequences and instructions within control flow

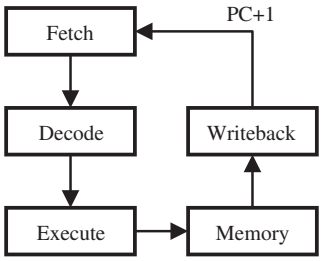

图 3 控制流指令执行流程

Figure 3 The execution process of instructions within control flow

中, 无时无刻都闪现着数据流计算思想的光辉.

本文首先回顾数据流计算思想、分析早期数据流计算机的局限性, 之后, 第 3 节分析在现代中央处理器 (central processing unit, CPU) 技术中所借鉴的数据流计算思想、乱序执行和多线程技术; 第 4 节介绍数据流计算思想在应用前沿下发生了新的变化, 流计算思想和流处理器技术; 第 5 节分析图形处理器 (graphics processing unit, GPU) 中所采用的数据流计算思想; 第 6 节从大数据分析系统角度分析数据流计算思想带来的新变化; 第 7 节分析智能计算加速器体系结构中数据流计算思想的运用. 最后总结数据流计算思想运用规律, 展望未来发展趋势.

2 数据流计算思想起源与早期数据流计算机

2.1 数据流计算思想起源

控制流计算思想在表达计算任务方面比较符合人类的思考习惯. 如图 2 所示, 首先通过指令序列描述计算任务执行过程. 其中, 指令用操作码表示功能, 操作数表示被处理的数据. 控制流计算任务被描述为一串指令序列. 经典计算机体系结构中, 设计了以程序计数器 (program counter, PC) 为核心的控制器实现控制流计算思想. 图 3 是基于控制流的指令执行流程图, 在执行时首先需要将待执行的指令取出, 经过译码后执行指令, 并通过地址访问内存, 获取需要的数据, 最后将结果数据写回到存储器中. 完成该条指令后, 程序计数器自动加一, 获取下一条指令继续上述流程. 在计算机系统发展初期, 研究人员逐渐发现控制流计算思想中存在的性能瓶颈问题, 即指令执行是串行的, 而且取指令、取操作数、存结果等动作都需要访问存储器.

在解决控制流计算的串行指令执行和访存瓶颈问题过程中, 在 20 世纪 70 年代前后, 麻省理工学院的 Dennis [1, 2] 提出了一种具有天生并行计算能力的数据流计算思想, 令人耳目一新. 数据流计算思想采用数据流图描述计算任务, 其中结点代表运算操作, 边代表所需要的数据, 如图 4 所示. 在基于数据流的计算任务中, 一项运算操作在获得了所有需要的操作数时即可执行, 所产生的结果数据不会被存储器保存, 而是直接作为操作数发送给后续的操作, 直到产生最终的输出结果. 与控制流计算思想中的指令序列串相比, 数据流图以图的形式描述了计算任务的并发执行全过程, 同时数据可以按照数据流图中边的指向直接传递到运算操作, 不需要缓存到存储器中. 这一创新思想立即在计算机体系结构界引起了轰动, 引发了多项基于数据流计算思想的计算机体系结构设计探索.

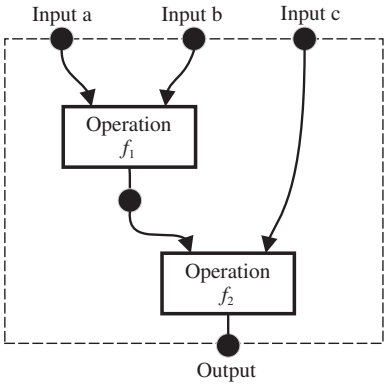

图 4　数据流示意图

Figure 4　The data flow diagram

2.2　早期数据流计算机

早期基于数据流计算思想的计算机系统试图完全按照数据流图的执行过程, 设计相关执行机构和相应运行方法, 这类机器被称为 “数据流计算机” 或 “数据流机”, 可分为静态数据流计算机和动态数据流计算机两类, 国内较早时候也有对数据流计算机进行介绍和归纳 [3, 4].

静态数据流机是根据数据流计算思想, 具体实现出来的一种执行机构, 其基本结构主要由取指令部件、指令存储部件、处理部件、更新部件等组成. 在静态数据流机中运行程序时, 操作数不通过 “寻址” 被访问, 而是通过 “令牌 (token)” 或 “值” 的形式 (图 4 中的实心圆点) 进行传递. 数据流计算机中的信息项由数据令牌和操作包组成, 其中, 数据令牌由结果值和目的地组成, 操作包由操作码、操作数和后继指令位置等组成. 待执行的数据流程序存放在指令存储部件中, 当某条指令所需的所有数据令牌都到达后, 取指令部件将相应的指令取出, 发送到可执行指令队列. 当物理计算资源有空闲时, 队列中的指令将依次被分配给处理部件进行并发执行. 这种等待结点所有输入弧中都获取到令牌后触发该结点执行的行为被称为 “点火 (firing)”, 是数据流计算模型的重要特点之一. 所产生结果形成新的令牌发送到更新部件中, 更新部件按令牌中的目标地址将令牌发送到指令存储部件内的相应指令位置. 如果有指令已经获取到了所有需要的令牌, 更新部件会将该指令的地址发送给取指令部件, 进行下一次的数据流执行. 麻省理工学院 (Massachusetts Institute of Technology, MIT) 的静态数据流机 [5] 是该类机器的典型代表.

动态数据流机重点解决数据流图的边存在多次执行的问题. 静态数据流机虽然具备基本的数据流执行能力, 但它不仅依赖于数据, 也需要配合带有简单控制信号的 “控制令牌” 来确认指令间的数据传送, 且无法保证并发的重复性调用 (例如递归) [6]. 对于这个问题, 动态数据流机通过 “标号令牌 (tagged-token)” 的方式, 使单条弧上可以同时存在多个不同的令牌. 相比于静态数据流机, 动态数据流机在执行机构上增加了一个 “匹配部件”, 该部件负责添加标号和匹配标号的工作, 使数据令牌的传送不再依赖于控制令牌. 更新/取值部件通过标号的配对, 从指令存储部件中获取指令, 结合收到的数据令牌组, 合并出可执行的指令, 送至可执行指令队列. 这样一来, 数据流程序的并行性得到了更大限度的开发, 也具备了更完善的功能. 动态数据流机主要可分为网络状 [7] 和环状 [6].

2.3　传统数据流计算机的局限性

从 20 世纪 70 年代中后期发起, 直至 80 年代末 90 年代初, 研究人员围绕数据流计算机展开了

丰富的研究探索, 研制了多种原型样机, 例如 Monsoon Machine [8], Manchester Data Flow Machine [9], J-Machine [10] 等, 但总体上都没有走向产业化道路. 总结传统数据流计算机的研制过程, 其固有的一些局限性, 导致其无法得到实用普及.

第 1 个限制因素是将传统的串行程序自动转换为数据流图所面临的挑战. 数据流计算思想的前提假设是将计算任务用数据流图描述出来, 但是大部分程序是程序员以串行程序语句描述的, 这些程序代码经过编译或解释后, 才能在与之兼容的执行机构上运行. 如果希望将这些程序运行在数据流计算机上, 则首先需要有工具将程序自动转换为对应的数据流图. 通过编译器将简单的科学计算核心算子描述为数据流图是可以做到的, 但是真实的大型应用程序或者大量事务处理程序, 包含了大量动态数据结构、复杂的条件判断等结构, 将其自动转换为数据流图仍然困难重重.

第 2 个限制因素是存储资源因素, 数据流程序没有数据共享机制, 数据量较大时会导致大量的数据复制, 无法进行局部性优化, 造成存储空间的浪费和大量的传输开销. 在数据流图中, 系统无法得知 token 需要多少资源, 仅能通过 token 的数量来进行粗略的资源估计, 难以准确分配和释放资源, 影响整体资源利用率. 这种过于简单的存储结构在 “数据到齐就执行” 的调度策略下, 经常会引起资源不足的问题 [11].

第 3 个因素是程序的编写和调试难度. 传统的数据流程序并不能完全实现控制流程序的所有功能, Dennis 在初步设计数据流编程语言时也指出了该语言在语义、内存管理等方面的不足 [2]. 另外在执行过程不确定的环境下调试数据流程序是一个很大的挑战, 即使在现在机器上使用的数据流编程语言, 在大数据时代, 由数据引起的错误在调试时通常也会花费较多的时间 [12].

虽然纯粹的数据流计算机没有得到产业化应用, 但是在体系结构探索过程中, 留下了很多值得借鉴的经验与教训. 一方面, 在赞叹数据流计算思想具有大规模并行表达能力的同时, 研究人员也认识到了数据流计算思想必须满足一定的前提条件, 即计算任务需要用数据流图描述. 既然自动生成复杂数据流图是当前不可逾越的一个障碍, 那么提供一个便捷的数据流图描述语言, 让程序员在宏观层面描述计算任务中的数据流关系, 为数据流计算思想的广泛运用带来了新的契机. 另一方面, 研究人员将数据流计算思想约束在局部范围内, 可以自动构建数据流图, 以及数据流计算机中采用的现场切换等技术, 都可以在控制流计算机中运用, 以提高其计算效率.

3 控制流计算机中的数据流计算思想

自 20 世纪 90 年代以来, 互补金属氧化物半导体 (complementary metal oxide semiconductor, CMOS) 大规模集成电路技术迅猛发展, 集成电路的集成度每 18 个月翻一番, 推动计算机系统技术进入快速发展时期, 此时计算机系统技术的核心体现为微处理器技术. 毫无疑问, 控制流计算思想主导着这些计算机系统的体系结构设计, 主流编程语言是串行程序设计语言, 既可以描述科学工程计算又可以描述事务处理应用, 因此也称为通用计算机、通用微处理器或简称 CPU. 在控制流计算思想占主导地位的同时, 数据流计算思想并没有销声匿迹, 其对计算效率提升的多种关键技术潜移默化地融汇到 CPU 设计的方方面面, 为 CPU 计算效率提升发挥了重要作用.

现代通用 CPU 指令流水线的设计体现了数据流计算思想. 指令流水线是 CPU 的控制核心, 在 CPU 设计中通过采用硬件对指令流片段进行数据依赖分析, 分支预测, 然后根据预测的数据值选择何时执行指令, 体现了操作数一旦就绪立即执行的数据流计算思想. 另外在指令流出控制过程中, 传统控制流计算思想是顺序流出指令, 现代高端 CPU 体系结构中采用了乱序执行技术, 其基本执行概念如图 5 所示. 乱序执行部件 [13] 在硬件译码阶段静态完成寄存器重命名, 建立保留站以存储不同类型

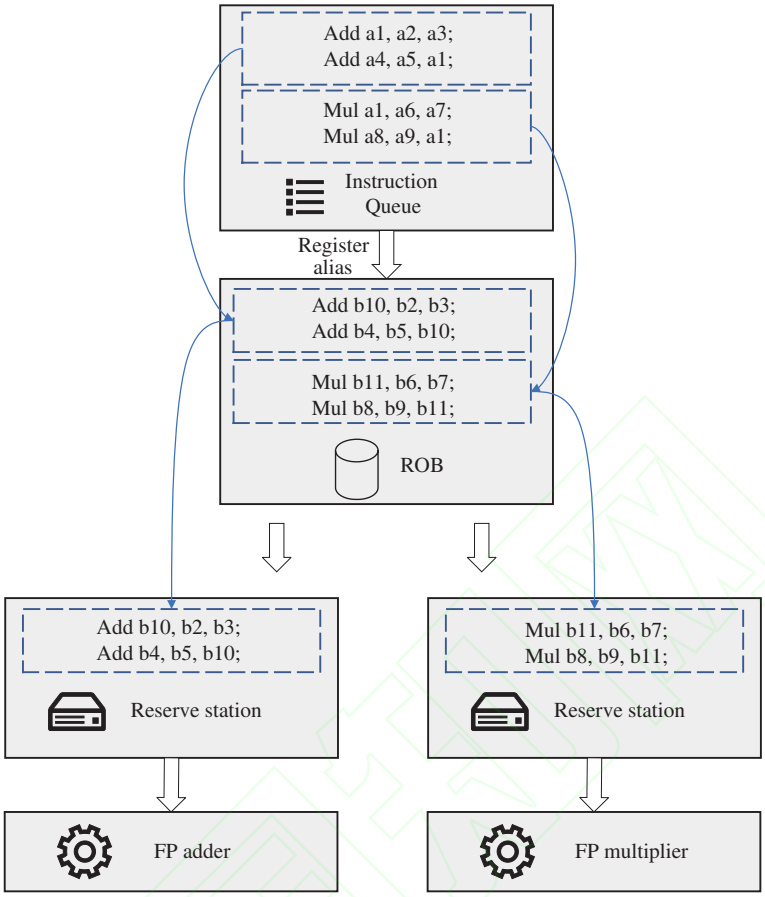

图 5　(网络版彩图) 乱序执行概念

Figure 5　(Color online) The concept of out-of-order execution

微指令, 同时监视公共数据总线输出结果, 一旦操作数就绪, 等待的微指令立即执行; 此外, 使用重排序缓冲区进行硬件推测, 使得不同微指令在不同处理单元可以同时全速运行, 达到一个指令周期并行执行 11 条微指令, 极大降低流水线停顿造成的性能损耗 [14]. 在上述技术运用中, 借鉴了数据流计算思想, 主要体现在预先判断数据就绪, 然后乱序执行. 通过限制数据流图范围, 在基本块层对数据相关性进行分析, 采用硬件自动判断数据相关, 在增加有限硬件代价的基础上, 达到了提高流水线效率的目的.

CPU 设计中的多线程技术也体现了数据流计算机的现场切换思想. 在现代 CPU 中多线程技术得到了广泛运用, 在 Intel 处理器中称为 Hyperthread 技术 [15], 在 AMD 处理器中称为 multi-threading technology 技术 [16], 在原 SUN 多核处理器中称为 SMT (simultaneous multithreading) 技术 [17]. 这些多线程技术在硬件上为每个处理器准备了多个现场寄存器 (context), 当线程执行了长延迟的操作, 处理器调度器可以动态将另一个处理器现场切换过来, 调度其他线程指令执行, 提高了硬件资源的利用率. 在编程层面需要程序设计人员给出相关描述, 程序设计人员为了提高应用程序的资源利用率、隐藏资源访问延迟, 通过对应用程序的数据依赖分析将其分解成多个模块, 使用线程调度的方式提高资源访问无关模块的并行性和访问相关模块的连续性, 合理的多线程调度可以使应用程序使用资源更加灵活.

4 数据流计算思想的嬗变: 流处理思想和流处理器

新应用需求为数据流计算思想的运用带来了新契机. 20 世纪 90 年代末, 个人计算机不断普及, 已经开始走进千家万户, 视频编解码、图像声音处理等多媒体处理需求日益强烈, 通用处理器在面对这类应用方面, 计算能力不能满足应用的需求. 例如, 在当时最先进的 586 台式计算机[1)]上播放 VCD 视频, 其解码速度不能满足每秒 25 帧的观看要求, 不得不配置 MPEG1 解码卡. 这类应用具有鲜明特点, 首先计算访存比高, 例如在视频编解码等应用中, 计算访存比可以高达 40 以上. 其次, 计算有固定规律, 可以并行处理, 适合利用单指令多数据流 (single instruction multiple data, SIMD) 等单元同时处理. 最后, 数据局部性强, 表现在数据会被多次使用, 以及多个功能模块之间构成生产者 – 消费者局域性, 即数据的流向是明确的、用户可控的. 这类应用的处理过程以数据为中心, 数据元素通常连续排列, 具有明显的数据持续流动计算特点. 面向这类新的应用, 以斯坦福大学 (Stanford University) Dally 教授为代表的一批学者提出了具有流处理思想的计算机体系结构.

流处理思想在编程方式上借鉴了数据流计算思想, 提出了 Stream-Kernel 流编程模型 [18]. 其主要思想是通过 Stream-Kernel 两级编程模式对数据的传递过程和计算过程进行分别描述. Stream 级程序其主要功能是将数据组织成流的形式, 一般运行在主处理器端, 负责在主处理器和流处理器之间进行数据的传输. Kernel 级程序用于描述数据被计算的过程, 类似串行语言的循环体. Kernel 程序运行在流处理器的计算簇上, 重复对输入数据进行处理, 并产生输出流, 用于后续 Kernel 计算. Kernel 程序在逻辑上仅描述数据流中一个单位数据的计算过程, Stream 程序描述多个数据的传递过程, 在执行时多个数据会被流处理器反复执行同一个 Kernel 程序. 这种分层的编程模式融合了数据流和控制流的优点. 在应用层面, 程序员通过将数据组织成流的形式, 用于描述具体应用中不同 Kernel 之间数据的传递过程. 在 Kernel 层面其假设对流入的数据进行相同的 Kernel 计算, 简化了对程序控制流的判断, 达到流水化处理效果. 该方法避免了直接从串行程序设计语言中提取数据流图的困境, 利用 Stream 描述数据流动过程, 利用 Kernel 描述可以被并行执行的循环体.

在 Stream-Kernel 流编程模型的基础上, 流处理器在结构上也实现了相应的硬件支持. 在结构上, 流处理器对数据访问、组织指令和运算操作指令 3 大模块进行解耦, 分为两大部分 [19]. 首先采用流调度模块, 通过索引和再定序机制将数据组织成流, 组织成流的数据排列规整, 可以在运算阵列上以 SIMD 或 MIMD (multipe instructions multiple data) 的方式并行执行. 其次, 流计算模块专注于密集计算, 支持大量运算单元阵列, 以获取极高的计算能力, 在具体实现上采用了 VLIW 指令格式, 简化指令流出的硬件开销. 最后, 为了满足大量计算单元对数据的需求, 流处理器采用了多级存储的形式, 由本地寄存器、流寄存器文件和动态随机存取存储器 (dynamic random access memory, DRAM) 三级存储空间构成, 其中流级存储器文件是程序员可管理的存储空间, 在访存速度上与 cache 相当, 主要存储 Kernel 要处理的临时数据, 即在 Kernel 被执行之前数据已经以流的形式加载到流寄存器文件, 满足大量运算单元对数据的快速访问需求.

典型的流处理器有 Imagine [19], RAW [20], Merrimac [21], MASA-1 [22] 等. 其中最具代表性的是 Imagine, 它是斯坦福大学 Dally 教授团队在 2002 研制的一款流处理器. Imagine 流处理器没有采用 Cache, 而是采用流寄存器文件 (stream register file, SRF) 作为流 (主) 存储器与处理器寄存器之间的缓冲存储器, 解决存储器带宽问题. 在 250 MHz 下, Imagine 原型系统在典型应用上可达到 10 GFlops 计算能力, 功耗仅为 6 W. 流处理器虽然没有在产业界得到广泛应用, 但是其数据流与控制流想结合的创新思想为后续体系结构创新开辟了一条新技术路线. 显然, 流处理的执行模式是将数据组成连续记

1) https://www.informit.com/articles/article.aspx?p=130978&seqNum=29.

录的格式, 称之为流, 通过流在不同 Kernel 之间的传递刻画数据的流动和执行过程, Kernel 利用大规模的计算阵列对流数据进行预先设置好的并行计算, 包括指令级、数据级和任务级的并行. 但是流处理和数据流还是有本质上的不同, 流处理本质上还是基于控制流思想, 是由指令驱动执行的. 它在局部借鉴了数据流并行的思想最大化利用流处理器的计算资源, 同时提供同步机制保证程序的正确性. 而数据流模型是一种异步并行的模型, 将本来同步执行的程序变成异步执行, 没有同步机制, 采用点火的方式由数据驱动程序的执行.

围绕数据流计算思想的研究在学术界仍然在持续进展, 以高光荣教授团队 [23] 为代表, 提出了数据流 Codelet 模型, 其主要思想是提升数据流图描述的粒度, 结点为一组可被原子性调度的机器指令, 其粒度介于控制流计算机的线程模型和数据流计算机之间. 进一步地, 引入了被称为 “线程程序 (threaded procedure, TP)” 的层次并行思想, 与早期的 EARTH 模型相似 [24]. 起源于数据流的 Codelet 编程模型与系统继承了数据流细粒度、高并行的特点, 将 “硬件指令” 改变到 “代码片段”, 采用避免了细粒度的上下文切换开销, 并以二级并行的设计增加了传统数据流所不具备的共享数据的局部性, 为程序员提供了更方便的接口. 尽管不是实现传统意义上的数据流计算机, 但 Codelet 模型为大规模计算系统提供了重要的借鉴意义. 特拉华大学 (University of Delaware) 的团队 [23, 25] 实现了 Codelet 的运行时系统 DARTS, 将模型中的 Codelets、线性程序、循环等元素准确地通过 C++ 在 x86 架构上实现出来, 并对它们的调度和点火激发等设置了各种对应的调度器, 用户可以通过 DARTS 在通用硬件上使用 Codelet 模型进行编程. 此外, 数据流的提出者 Dennis 教授和 Codelet 的提出者高光荣教授进行合作, 基于 Codelet 模型, 实现了 Fresh Breeze 架构 [26, 27]. 还有一些工作 [28, 29], 虽然没有直接实现 Codelet 模型, 但其设计思想也是深受 Codelet 的启发, 或与其相关. 而对于 Codelet 模型在处理迭代任务上的资源利用率不足的问题, 中国科学院的团队设计了一款面向科学应用的数据流加速器 [30], 以分离迭代和主循环的方式, 使数据流图可以更独立地执行, 并在多个层面提升并行度.

5 数据流计算思想在 GPU 产业化中落地应用

进入 21 世纪, 处理器性能增长遭遇时钟频率提升的瓶颈, 计算机体系结构研究开始进入探索多核并行计算体系结构时期. Intel 和 AMD 在 2006 年前后相继推出多核处理器系列芯片, 正式开启多核处理器产业化时代. 虽然暂时保持了处理器性能持续提升的势头, 但是, 面对应用领域对计算性能数量级跃升的需求, 采用通用处理器集成多核的技术路线前景依然不容乐观, 此时异构并行技术路线脱颖而出.

GPU 作为 CPU 附属的专用显示处理器件, 集成了大量计算单元, 成为显著提升计算性能的有力竞争者. GPU 图形处理加速器初始只是作为一种辅助计算设备存在, 用于图形显示的像素光栅化, 配合高端工作站完成图形显示任务. 2000 年后英伟达 (NVIDIA) 在其 GPU 中引入了 “T&L” 硬件, 在硬件上具备了顶点变换能力, 可以独立完成图形显示任务. 此后, 随着应用对图形显示能力的需求日益强烈, GPU 硬件开始集成可编程顶点处理器和可编程像素处理器. 这一时期, GPU 的编程模式是采用面向图形显示的专用编程工具, 例如: DirectX [31], OpenGL [32] 和 Cg [33], 编程效率不高, 难以使用. 2006 年之后, GPU 开始从单纯的图形渲染领域转向通用计算领域, 也称为 GPGPU (general-purpose computing on graphics processing units). 通用计算本来是传统 CPU 的主导应用领域, GPU 要在这个领域有所斩获, 需要在编程模型和硬件架构上都有所突破, 必须兼顾效率和易编程的特性.

流编程模型为 GPU 成为通用计算平台发挥了关键作用. 为了满足通用应用在灵活性和高性能方面的需求, GPU 的 CUDA [34] 编程模型也采用了数据流的思想, 继承并改善了传统流处理器中的

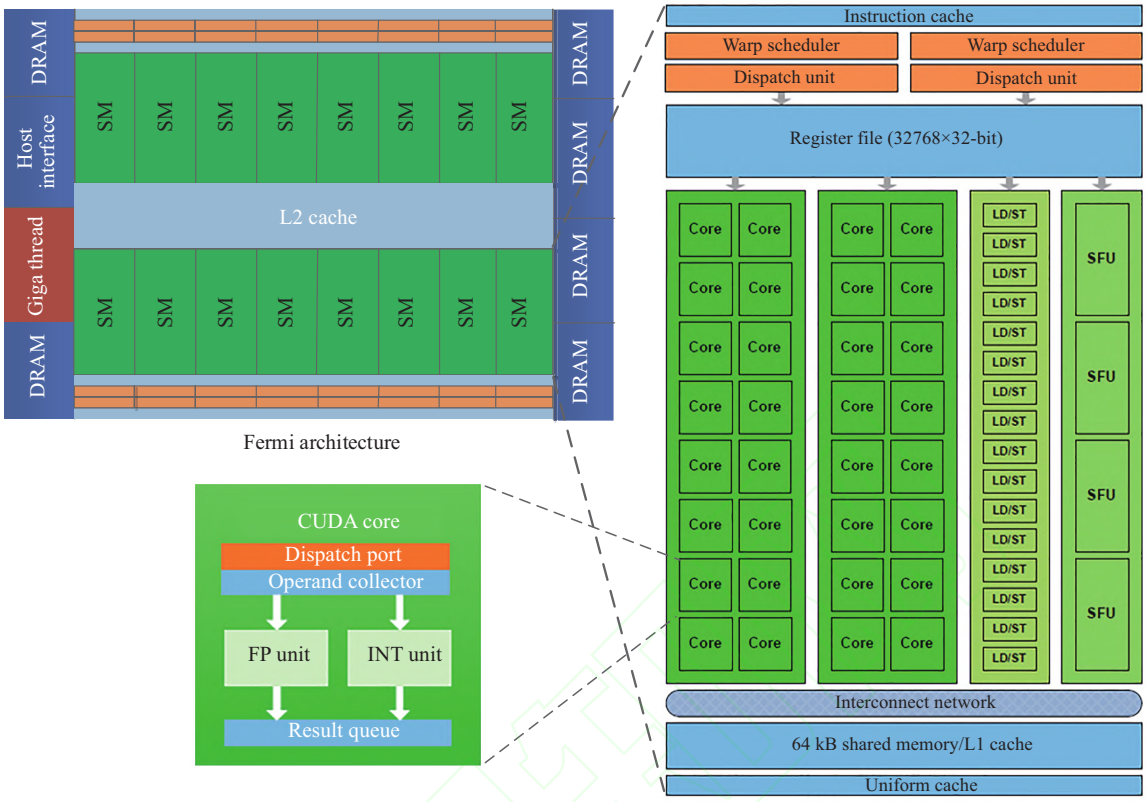

图 6 (网络版彩图) Fermi 体系结构

Figure 6 (Color online) The architecture of Fermi GPU

Stream-Kernel 编程模型. 首先, 现代 GPU 编程继承了流编程模型中的多级并行编程方式, 采用 Kernel 描述通用应用中数据密集的算法, 提高其在硬件执行上的效率. Kernel 将相关的指令集成到一个执行节点中, 构成一个大的数据流执行节点, 不同的 Kernel 以异步的方式执行, 其执行顺序由数据是否就绪和资源是否满足决定, 并不受限于其在程序中的顺序决定, 具有明显的数据流计算特征. 其次, 为了满足通用应用不同特性的需求, GPU 编程采用了更为灵活的 MIMD 执行模式, 而不是严格的 SIMD 运行方式, 其并行度可以达到上万级别. 每一个线程就像一个自带标签的数据流计算节点, 只要数据和资源就绪, 线程即可执行, 完成其任务之后就释放资源.

以流处理器为基本计算单元, GPU 为通用计算领域计算性能提升开辟了新的空间. GPU 通过堆叠大量的流处理单元满足数据流计算程序中丰富并行性对数据流计算节点的需求. 以 NVIDIA GPU [35] 为例, 它由众多的流多处理器 (streaming multiprocessor, SM) 构成, 每一个流多处理器又由简单的流处理单元组成并以 SIMD 的方式执行, 每一个流处理单元都可以看成是一个数据流执行节点, 不同的 SM 之间以相互独立的方式运行, 共同实现 GPU 灵活的 MIMD 执行模式. 在微观层面, 数据的处理由不同的线程完成, 每个数据都由线程 ID 对应, 类似于数据流中的带标签的 token, 线程以乱序的方式执行, 只要线程获得了资源 (包括计算单元、寄存器) 即可执行相应的指令流, 极大地提高了并行效率, 其基本结构如图 6 所示. 以大量本地寄存器和软件可管理的片上存储为核心的多级存储结构, 为 GPU 中的数据流动提供了极高的带宽, 满足大量流处理单元对数据的访问需求.

GPU 并不是完全意义上的数据流处理器, 它结合控制流和数据流计算的思想, 在计算性能和易编

程性上实现了较好的平衡, 以极高的计算性能满足了通用计算的需求. 依托 CUDA 这种数据流与控制流结合的编程工具, 英伟达构建了庞大的 GPU 生态系统, 在学术界和产业界广泛使用. 在高性能计算领域, CUDA 提供了 cuBlas, Thrust 和 cuFFT 等高性能计算核心库. 在深度学习领域, cuDNN 卷积优化库已经被几乎所有的深度学习框架所囊括, 作为后端加速库支撑深度学习框架在 GPU 上高效运行. 当前英伟达主流 GPU 产品, 以绝对主导地位占据消费类游戏等桌面显示市场; 在数据中心和典型高性能计算场景领域, 目前大部分采用 CPU+GPU 异构并行架构. 除此之外, 以 Jetson TX2 为代表的嵌入式 GPU 在自动驾驶等嵌入式领域也获得广泛应用. GPU 将数据流计算思想与通用控制流计算在编程语言、存储层次设计、微体系结构设计等多个方面融会贯通, 取得了学术与产业双丰收, 成为计算机体系结构平衡设计的经典范例.

6 数据流计算思想在云计算大数据时代的发展

2005 年后随着 CPU 进入多核时代, 计算机系统的硬件基础逐步趋于稳定, 计算资源呈现集中化的趋势, 云计算系统成为计算机系统的新形态, 网络通信基础设施的逐步完善是计算资源集中使用的必要条件. 随着信息技术和网络技术的快速发展, 为了解决服务器集群在低负载时产生的资源浪费和闲置, 云计算技术将具体业务与基础设施维护分离. 云计算将网络服务资源虚拟化, 使得计算资源可以按需使用. 更进一步, 云计算从服务角度可分别在基础设施 (Iaas)、平台 (Paas)、软件 (Saas) 3 个维度使得用户可以便利地使用虚拟资源处理计算问题.

与此同时, 网络通信技术快速发展促进了人机物信息空间与物理世界高度融合, 大数据处理成为时代的焦点. 高速骨干网将数据中心、服务器等计算资源连接起来, 移动互联网通过移动终端将人类社会连接起来, 物联网将传感器等机械装置连接起来, 构成了万物互联的信息物理空间. 毫无疑问, 这个信息物理空间所依托的载体就是数据, 它是这个空间的客观反映, 每时每刻都在记录, 每时每刻都在变化. 人们用大数据摩尔定律 (Moore's law) 来描述其增长态势, 即全球数据以每年 50% 的速度增长, 也就是说每两年就增长一倍. 围绕大数据技术应用, 一批新兴的数据挖掘、数据存储、数据处理与分析技术不断涌现, 对计算机系统提出了新的要求.

数据流计算思想给云计算与大数据分析系统领域的编程模型带来了重大变化. 可以看到, 大数据处理系统的编程模型都倾向于主体采用数据流描述方式, 在计算层以函数形式采用串行程序设计语言描述. 大数据处理系统的执行机构主要是分布式集群计算系统, 与云计算系统相互融合发展. 图 7 展示了数据流思想在经典或常用的数据处理系统和编程模型上的发展过程和趋势, 本节将依次介绍图中 4 种计算模型和其中的数据流思想.

6.1 适应批数据处理的 MapReduce 编程模型与系统

较早的大数据处理系统是从基于 MapReduce[36] 编程模型的大数据批处理计算框架开始的. 该编程模型将计算描述成 Map 和 Reduce 两个阶段, 分别对输入数据进行解析和规约等操作, 其中 Map 操作表达一组数据进行并行处理, 其算子用户可以自己定义, 类似流编程模型中的 Kernel. 宏观上看, 这两个主要阶段之间存在数据驱动关系, 即 Reduce 节点需要在获得 Map 节点生成的含有特定 key 的键值对以后, 再对它们进行 Reduce 操作. 从这一角度来看, MapReduce 编程模型是适应大数据批处理的数据流模型. 其本质上是让程序员显式地描述数据处理流程, 是描述数据流图的一种简化形式. 这个编程模型虽然简单, 但是非常适合描述对批量大数据进行并行处理. MapReduce 作为大数据时代早期的代表性系统之一, 为大数据处理系统的发展做出了重要的贡献: (1) 编程方法简单, 隐藏了底层复

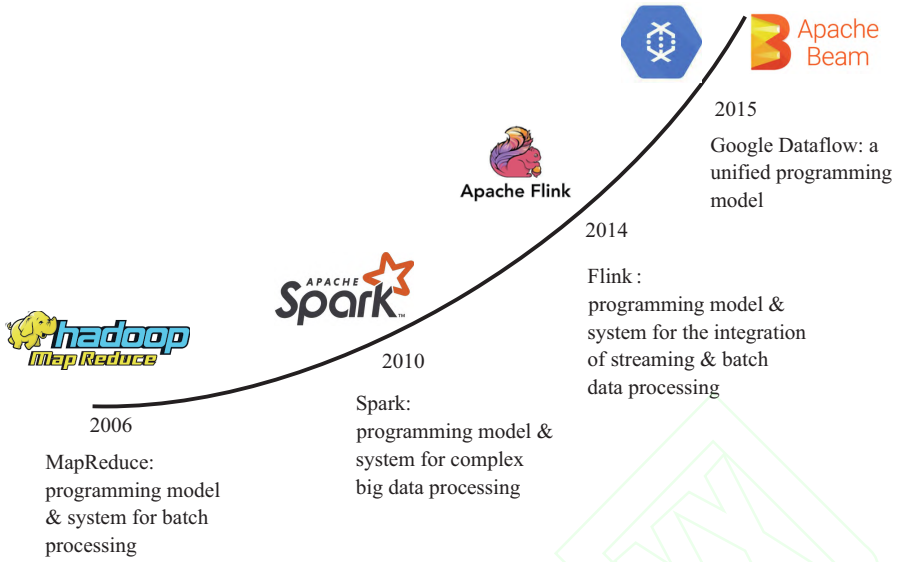

图 7 (网络版彩图) 数据流思想在常用计算模型中的应用

Figure 7 (Color online) The application of data-flow computing on several programming models

杂的处理细节, 用 Map 和 Reduce 就可以实现批量数据的并行处理; (2) 适用于大规模的分布式计算集群, 降低了对单机配置的要求, 具有良好的扩展性; (3) 考虑到分布式环境下的机器故障率, 实现了高容错性等. 然而, MapReduce 也存在一些不足: 受 Map 和 Reduce 步骤的制约, 其编程模型的表达能力有限; 在系统实现上, 频繁读写磁盘导致 I/O 开销较大, 不适用于低延时和重复迭代的计算任务等. 值得指出的是, 虽然 MapReduce 在宏观的编程模型上可以视为数据流模型, 但在微观的系统实现上它仍然使用的是控制流模型.

6.2 适应复杂大数据处理模式的 SPARK 编程模型与系统

在 2010 年前后, MapReduce 已经发展成一个常用的大数据分布式计算框架, 但由于 6.1 小节中提到的这些限制, 使其逐渐无法满足日益丰富的应用需求. 此时, Spark [37] 作为一个开源分布式计算框架诞生了. Spark 提出了基于弹性分布式数据集 (resilient distributed dataset, RDD) 数据抽象的数据流编程模型, 提供多种转换算子 (transformations) 用于描述数据流动的多种形式. Spark 编程模型支持程序员描述数据流图中的边, 其结点是 RDD 数据对象及相关计算, 属于典型的数据流编程模式. 在执行过程中, Spark 采用类似数据令牌 (token) 机制, 启动各个操作节点计算驱动数据流动, 每经过一个操作节点可产生新的 RDD 供后续节点使用. 值得指出的是, 这些操作可在内存中完成, 避免中间数据的磁盘 I/O 操作, 直到产生最后输出时才进行持久化存储.

Spark 计算模型的优点有: (1) 为 RDD 的操作流程提供了丰富的编程接口, 比 MapReduce 更强的表达能力; (2) 内存型的执行引擎显著降低了 I/O 开销, 提升了系统整体性能; (3) 基于 RDD 和数据流模型的 “血统 (lineage)” 重算机制, 为快速的容错提供了保障等. 不足之处有: (1) 执行引擎的批处理模式无法充分满足高实时性的需求; (2) 长时间处理迭代任务时, 由于内存容量有限, JVM 中的中间数据堆积会导致 Spark 的性能不稳定 [38], 因而存储管理也是数据流模型都要面对的一个挑战 [2, 39].

6.3 适应批流融合应用新需求的 Flink 编程模型与系统

2014 年, 大数据的发展达到了一个高峰, 虽然那时批数据仍然普遍存在, 但流数据的处理需求开始在诸如通信、交易等场景中占据重要地位. 在这种批处理和流处理的需求同时存在、流处理的重要性不断增加的背景下, Flink [40] 作为一种低延时、高容错的分布式流处理框架诞生了. Flink 是真正意义上的 "流处理引擎", 数据进入后立即被处理, 更符合数据流模型的 "点火规则" —— 令牌到齐后立即激发执行. Flink 的另一个特点是 "异步屏障快照" 容错机制, 该机制中的检查点屏障可以融入数据输入流中, 它像普通的令牌一样在数据流图的节点间流动. Flink 的优点在于: (1) 流式处理引擎满足低延时的需求; (2) 将批式数据视为有限的流, 交由流处理引擎统一处理, 使用户不需要同时编写批流两种模式的代码; (3) 异步屏障快照的容错机制降低了容错的额外开销等. 其缺点在于 [38]: (1) 内存依然是限制因素, 若设置不当则会影响性能和资源利用率; (2) 在分布式执行迭代任务时, 对网络资源的占用较高; (3) 大图处理等复杂的、多过滤层的任务的处理性能相对低于 Spark.

6.4 理想化的 Google Dataflow 编程模型

谷歌于 2015 年提出 Dataflow 模型 [41], 为分布式海量数据处理提供一个更简单灵活的编程模型, 满足批流融合、低延时、高容错, 以及良好的表达能力的需求, 其开源版本为 Beam[2)]. Dataflow 模型对数据类型提出了更明确的划分方法: 输入数据依据是否有限可分为有界的 (bounded) 和无界的 (unbounded), 且有界是无界的一种特例; 批 (batch) 和流 (stream) 则用于划分执行引擎的类别. 这样的划分方法避免了编程模型层和执行引擎层在概念上混淆. 与传统数据流模型相似, Dataflow 模型描述了 "数据在节点间流动和变化" 的过程. 不同之处在于, Dataflow 作为编程模型不涉及执行层面的实现细节, 而传统数据流 [5] 则是执行模型. 为了使用户可以更高效地描述各种场景下的数据流计算任务, Dataflow 编程模型提供了事件时间、窗口模型、触发器等工具, 这也为 Spark, Flink 等系统后续的演进提供了重要参考. Dataflow 模型的优点有: (1) 丰富的编程接口, 提高了模型的表达能力和用户的编程效率; (2) 将上层编程接口与下层执行引擎解耦, 同样的代码可支持不同的执行引擎, 为用户提供更大的选择空间, 从而满足更丰富的应用需求; (3) 提供自动的代码优化和资源管理功能等. 其缺点有 [42]: (1) 增加了下层执行引擎的适配要求; (2) 为支持更多特性 (如带状态的处理), 引入了额外的空间和计算开销; (3) 开放过多接口给用户增加了用户编写程序的学习成本等.

7 智能算法加速器时代的数据流计算思想

2015 年以来, 在大数据、强算力的支撑下, 以深度卷积神经网络 [43~45] 为代表的深度学习算法成为人工智能领域主流方法, 广泛应用于图像 [46~48]、视频 [49]、声音 [50~52]、文本 [53] 等感知智能领域, 取得了显著应用效果. 典型案例是 2012 年以来采用深度卷积神经网络的 Alexnet [43], VGG [54], GoogleNet [46], Resnet [55] 等深度学习算法, 在 ImageNet 图像分类比赛 [56] 中超越传统机器学习算法, 显著提升了分类准确度; 另一个典型案例是 2016 年 DeepMind 公司采用深度学习结合搜索算法的游戏程序在围棋比赛中完胜人类顶尖高手 [57], 引发了人工智能领域的研究热潮.

面向深度学习算法开发定制加速器成为计算机体系结构设计的新热点. 学术界以中国科学院计算技术研究所陈云霁 [58, 59] 的 "DianNao" 系列学术成果为代表, 从算法、算力以及数据访存优化的角度提出了多种深度学习加速器体系结构, 近年来深度学习加速器成为计算机体系结构会议的热点话题.

2) Apache Beam. https://beam.apache.org/.

商业化 AI 芯片有谷歌的 TPU (张量处理单元)[3], 中国科学院计算技术研究所的寒武纪 [4], 以及华为的 atlas[5] 等; 随着各类定制芯片及 AI 加速器的提出, 其迅速应用于智能驾驶、搜索引擎、语音识别等多个领域, 比如地平线的面向智能驾驶的 “征程” 系列处理器 [6] 和面向 AIoT 的 “旭日” 系列处理器 [7], 已大规模商用, 而华为的麒麟及升腾芯片 [8] 也广泛应用在了手机端.

数据流计算思想在智能计算定制加速器体系结构设计中体现在面向神经网络的数据流图编程方法方面. 通过深度学习开发框架表达神经网络数据流图成为连接智能计算定制加速器和人工智能应用之间的紧密纽带. 以 TensorFlow [60] 主流深度学习开发框架为例, 在该框架下程序员显式描述神经网络的卷积层、池化层、非线性映射、全连接层等算子及其之间的连接关系, 构建数据流图. 算法加速器厂商针对算子开发高效算子库, 这些算子库对应用开发者屏蔽了编写定制加速器代码细节, 定制加速器设计者只需要通过提供算子库的形式, 就可以支持大量神经网络应用程序开发. 其中数据流图成为了支持硬件多样性的中间层, 同时系统软件可以按照数据流图中的数据传递关系动态调度算子执行, 支持应用程序在多种算法加速器上高效执行. 数据流图编程模式适合深度神经网络有限种类算子, 算子被多次重用的特点, 通过应用开发人员直接描述数据流图简化了应用开发和面向定制加速器体系结构定制优化程序的难度.

在智能计算定制加速器芯片结构设计中, 数据流思想更发挥了重要作用. 以深度学习中最具代表性的卷积神经网络为例, 在卷积神经网络硬件设计中有 4 个层次体现数据流计算思想, 分别是层级、特征图级、窗口级和操作级. 层级是指相邻卷积层之间仅存在写后读相关, 后层计算仅依赖前一层的结果, 而前一层不依赖于后者, 因此在数据流图上表现为有向无环图, 根据这一特点可以设计卷积层之间的运算流水线 [61], 直接传递数据, 避免存储开销. 特征图级是指卷积操作中不同输入通道、输出通道数据均不存在数据相关, 可以设计并行通道. 窗口级是指卷积算子在当前特征图上滑动时, 前后滑动的位置有部分重叠, 存在只读数据相关, 可以利用其减少访存操作. 操作级则是指卷积算子内部的元素之间实现点积累加操作, 可以利用数据流图构造累加树, 提高操作并行性. 因此, 数据流计算思想通过数据流图所表达的核心要义在定制加速器部件设计中处处得以体现.

8 总结与展望

本文从计算机体系结构平衡设计的视角分析了数据流计算思想在计算机系统发展历程中所发挥的作用. 传统数据流计算机在选择编程语言方面试图直接将串行语言转变为数据流图, 这个在当今时代仍然是巨大挑战的复杂问题导致其无法走向实用. 反之, 在 GPU、大数据集群系统中, 系统设计选择类似数据流编程语言作为起点, 在应用开发人员能够接受的同时, 降低了研发编译器难度, 同时配合面向特定应用领域库函数和系统管理功能上的支持, 取得了成功. 展望未来发展, 在大规模集成电路技术没有颠覆性变化的前提下, 计算机体系结构进入稳定发展时期, 突出表现为应用创新牵引系统发展. 类似深度学习应用带来了智能计算算法加速器体系结构的发展, 智能应用将进一步深化, 从感知走向认知与决策领域, 应用对计算能力的需求依然强劲. 数据流计算思想在编程模型层面将进一步与特定应用需求紧密结合, 新型编程模型或编程框架会不断涌现, 例如在数据流图基础上描述用户对流

3) https://analyticsindiamag.com/tpu-beginners-guide-google/.
4) https://www.welcome.ai/cambricon.
5) https://e.huawei.com/cn/products/cloud-computing-dc/atla.
6) https://www.horizon.ai/product/journey.
7) https://www.horizon.ai/product/sunrise.
8) https://www.huaweicloud.com/ascend/home.html.

数据处理延迟或带宽的需求, 或者是智能编程框架分化出专门支持自动训练的框架和支持强化学习的框架. 在计算机系统层面, 与特定应用领域需求紧密结合的计算机体系结构将进一步丰富多彩, 在集成电路新工艺和敏捷芯片开发技术的支持下将走向黄金时代.

参考文献

1 Dennis J B. Programming generality, parallelism and computer architecture. In: Proceedings of International Federation for Information Processing Congress, Edinburgh, 1968. 484–492
2 Dennis J B. First version of a data flow procedure language. In: Proceedings of Symposium on Programming, Berlin, 1974. 362–376
3 Lu X D, Weng C L. Computer Architecture. Beijing: High Education Press, 2008 [陆鑫达, 翁楚良. 计算机系统结构. 北京: 高等教育出版社, 2008]
4 Li G J. A new kind of computer architecture-data flow computer. Comput Rev, 1981, 11: 1–8 [李国杰. 一种新的体系结构 —— 数据流计算机. 电子计算机动态, 1981, 11: 1–8]
5 Dennis J B, David P M. A preliminary architecture for a basic data-flow processor. In: Proceedings of the 2nd Annual Symposium on Computer Architecture, 1974. 126–132
6 Gurd J R, Kirkham C C, Watson I. The Manchester prototype dataflow computer. Commun ACM, 1985, 28: 34–52
7 Arvind , Nikhil R S. Executing a program on the MIT tagged-token dataflow architecture. IEEE Trans Comput, 1990, 39: 300–318
8 Papadopoulos G M, Culler D. Monsoon: an explicit token-store architecture. In: Proceedings of the International Symposium on Computer Architecture, Washington, 1990. 82–91
9 Gurd J R. The Manchester dataflow machine. Comput Phys Commun, 1985, 37: 49–62
10 Dally W J, Chien A A, Fiske S, et al. The J-Machine: a fine grain concurrent computer. In: Proceedings of International Federation for Information Processing Congress, San Francisco, 1989. 1147–1153
11 Culler D E, Arvind. Resource requirements of dataflow programs. In: Proceedings of the International Symposium on Computer Architecture, 1988. 141–150
12 Contrerasrojas B, Quianeruiz J, Kaoudi Z, et al. TagSniff: simplified big data debugging for dataflow jobs. In: Proceedings of the Symposium on Cloud Computing, Chaminade, 2019. 453–464
13 Smith J E, Pleszkun A R. Implementation of precise interrupts in pipelined processors. In: Proceedings of the International Symposium on Computer Architecture, Boston, 1985. 36–44
14 Hennessy J L, Patterson D A. Computer Architecture: A Quantitative Approach. 2013, 1: 135–172
15 Marr D T, Binns F, Hill D L, et al. Hyper-threading technology architecture and microarchitecture. Intel Technol J, 2002, 6: 1–12
16 Chen J, Watson-III W A. Multi-threading performance on commodity multi-core processors. In: Proceedings of the 9th International Conference on High Performance Computing in Asia Pacific Region, 2007
17 Tullsen D M, Eggers S J, Levy H M, et al. Simultaneous multithreading: maximizing on-chip parallelism. In: Proceedings of The International Symposium on Computer Architecture, Santa Margherita Ligure, 1995. 392–403
18 Mattson P, Dally W J. A programming system for the imagine media processor. Dissertation for Ph.D. Degree. Stanford: Stanford University, 2002
19 Khailany B, Dally W J, Kapasi U J, et al. Imagine: media processing with streams. IEEE Micro, 2001, 21: 35–46
20 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 the International Symposium on Computer Architecture, Munich, 2004. 2–13
21 Dally W J, Labonte F, Das A, et al. Merrimac: supercomputing with streams. In: Proceedings of the Conference on High Performance Computing (supercomputing), Phoenix AZ, 2003. 35–35
22 Zhang C Y. Implementation of the MASA-I stream processor on FPGA. Comput Eng Sci, 2008, 30: 114–118
23 Suettlerlein J, Zuckerman S, Gao G R, et al. An implementation of the Codelet model. In: Proceedings of the International Conference on Parallel Processing, Lyon, 2013. 633–644
24 Theobald K B. Earth: an efficient architecture for running threads. Dissertation for Ph.D. Degree. Montreal: McGill University, 1999
25 Suetterlein J. Darts: a runtime based on the Codelet execution model. Dissertation for Master's Degree. Newark:

University of Delaware, 2014
26 Dennis J B, Gao G R, Meng X X. Experiments with the fresh breeze tree-based memory model. Comput Sci Res Dev, 2011, 26: 325–337
27 Dennis J B. Compiling fresh breeze codelets. In: Proceedings of Programming Models and Applications on Multicores and Manycores, 2014. 51–60
28 Mattson T G, Cledat R, Cavé V, et al. The open community runtime: a runtime system for extreme scale computing. In: Proceedings of IEEE High Performance Extreme Computing Conference, 2016. 1–7
29 Gao G R, Sterling T, Stevens R, et al. Parallex: a study of a new parallel computation model. In: Proceedings of IEEE International Parallel and Distributed Processing Symposium, 2007. 1–6
30 Ye X, Tan X, Wu M, et al. An efficient dataflow accelerator for scientific applications. Future Generation Comput Syst, 2020, 112: 580–588
31 Gray K. Microsoft DirectX 9 programmable graphics pipeline. Microsoft Pr, 2003. https://www.amazon.ca/Microsoft-DirectX-Programmable-Graphics-Pipeline/dp/0735616531
32 Kessenich J, Baldwin D, Rost R. The OpenGL shading language. http://www.opengl.org/documentation/glsl
33 Mark W R, Glanville R S, Akeley K, et al. Cg: a system for programming graphics hardware in a C-like language. In: Proceedings of the ACM SIGGRAPH, New York, 2003. 896–907
34 Nvidia C. Nvidia CUDA C programming guide. Nvidia Corporation, 2011
35 Fermi N. NVIDIA's next generation CUDA Compute Architecture, Fermi. 2009
36 Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Commun ACM, 2008, 51: 107–113
37 Zaharia M, Chowdhury M, Franklin M J, et al. Spark: cluster computing with working sets. In: Proceedings of the IEEE International Conference on Cloud Computing Technology and Science, Indianapolis, 2010. 10
38 Marcu O, Costan A, Antoniu G, et al. Spark versus flink: understanding performance in big data analytics frameworks. In: Proceedings of the International Conference on Cluster Computing, Taiwan, 2016. 433–442
39 David E C, Klaus E S, Thorsten V E. Two fundamental limits on dataflow multiprocessing. In: Proceedings of the Architecture and Compilation Techniques for Medium and Fine Grain Parallelism, Orlando, 1993. 153–164
40 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
41 Tyler A, Robert B, Craig C, et al. The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. In: Proceedings of the VLDB Endowment, Hawaii, 2015. 1792–1803
42 Li S, Gerver P, MacMillan J, et al. Challenges and experiences in building an efficient apache beam runner for IBM streams. Proc VLDB Endow, 2018, 11: 1742–1754
43 Krizhevsky A, Sutskever I, Hinton G E, et al. ImageNet classification with deep convolutional neural networks. In: Proceedings of the Neural Information Processing Systems, Nevada, 2012. 1097–1105
44 Kim Y. Convolutional neural networks for sentence classification. 2014. ArXiv: 1408.5882
45 Sainath T N, Mohamed A, Kingsbury B, et al. Deep convolutional neural networks for LVCSR. In: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, 2013. 8614–8618
46 Szegedy C, Liu W, Jia Y, et al. Going deeper with convolutions. In: Proceedings of the Computer Vision and Pattern Recognition, Boston, 2015. 1–9
47 Girshick R, Donahue J, Darrell T, et al. Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the Computer Vision and Pattern Recognition, Ohio, 2014. 580–587
48 Razavian A S, Azizpour H, Sullivan J, et al. CNN features off-the-shelf: an astounding baseline for recognition. In: Proceedings of the Computer Vision and Pattern Recognition, Ohio, 2014. 512–519
49 Fan Y, Lu X, Li D, et al. Video-based emotion recognition using CNN-RNN and C3D hybrid networks. In: Proceedings of the 18th ACM International Conference on Multimodal Interaction, 2016. 445–450
50 Yu D, Deng L. Automatic Speech Recognition: A Deep Learning Approach. Berlin: Springer, 2016
51 Ravanelli M, Parcollet T, Bengio Y, et al. The Pytorch-kaldi speech recognition toolkit. In: Proceedings of the International Conference on Acoustics Speech and Signal Processing, Brighton, 2019. 6465–6469
52 Zhang Y, Pezeshki M, Brakel P, et al. Towards end-to-end speech recognition with deep convolutional neural networks. 2017. ArXiv: 1701.02720

53 Lai S, Xu L, Liu K, et al. Recurrent convolutional neural networks for text classification. In: Proceedings of the 29th AAAI Conference on Artificial Intelligence, Texas, 2015. 2267–2273
54 Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. In: Proceedings of the Computer Vision and Pattern Recognition, Ohio, 2014
55 He K, Zhang X, Ren S, et al. Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, 2016. 770–778
56 Deng J, Berg A, Satheesh S, et al. ILSVRC-2012. 2012. http://www.image-net.org/challenges/LSVRC
57 Chang H S, Fu M C, Hu J, et al. Google deep mind's AlphaGo. OR/MS Today, 2016, 43: 24–29
58 Chen T, Du Z, Sun N, et al. DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning. SIGARCH Comput Archit News, 2014, 42: 269–284
59 Chen Y, Chen T, Xu Z, et al. DianNao family: energy-efficient hardware accelerators for machine learning. Commun ACM, 2016, 59: 105–112
60 Abadi M, Agarwal A, Barham P, et al. Tensorflow: large-scale machine learning on heterogeneous distributed systems. 2015. ArXiv: 1603.04467
61 Liu Z, Dou Y, Jiang J, et al. Throughput-optimized FPGA accelerator for deep convolutional neural networks. ACM Trans Reconfigurable Technol Syst, 2017, 10: 1–23

Reviewing data flow computing thinking from the development of computer architecture

Yong DOU[1], Jialun WANG[2], Huayou SU[1*], Chen XU[2], Xiaoli GONG[3], Wangdong YANG[4], Chuliang WENG[2], Zhanhuai LI[5], Kenli LI[4], Ge YU[6] & Aoying ZHOU[2]

1. *College of Computer, National University of Defense and Technology, Changsha* 410073*, China*;
2. *School of Data Science and Engineering, East China Normal University, Shanghai* 200062*, China*;
3. *College of Cyber Science, Nankai University, Tianjian* 300071*, China*;
4. *College of Computer Science and Electronic Engineering, Hunan University, Changsha* 410082*, China*;
5. *School of Computer Science, Northwestern Polytechnical, Xi'an* 710072*, China*;
6. *School of Computer Science and Engineering, Northeastern University, Shenyang* 110169*, China*
* Corresponding author. E-mail: shyou@nudt.edu.cn

Abstract In the development of computer architecture, the von Neumann computer architecture has been the mainstream architecture of computer systems. When it comes to Non-von Neumann computer architecture, data flow computer systems are undoubtedly the most mentioned. In this paper, from the perspective of the development of computer architecture, we analyze the important role of data flow computing thinking in both innovation and development of computer architecture. We first review the data flow computing thinking and analyze the limitations in its early stages. Then we analyze out-of-order execution and multi-threading technology, two important technologies of data flow computing thinking used in modern CPUs. We further introduce streaming computing, streaming processor architecture and the modern GPUs from the view of data flow computing thinking. Moreover, we analyze how to apply data flow computing thinking to the development of computer systems in the era of big data intelligence. Finally, we summarise the rules in data flow computing thinking and look forward to future development trends.

Keywords data flow, big data, hybrid computing, GPU, intelligent computing

Yong DOU was born in 1966. He is a professor of at College of Computer, National University of Defense and Technology. He is a senior membership of China Computer Federation. His research interests include high performance computing, reconfigurable computing, deep learning, and data mining.

Jialun WANG is a doctoral student in School of Data Science and Engineering, East China Normal University. He received his bachelor degree in computer science from Sichuan University. His research interests include parallel and distributed systems, in-memory computing, and data management.

Huayou SU was born in 1985. He is a Ph.D. recipient, an assistant professor as College of Computer, National University of Defense and Technology. He is a member of China Computer Federation. His research interests include high performance computing, parallel computing, and CPU-GPU hybrid computing.

Aoying ZHOU is a professor at School of Data Science and Engineering (DaSE), East China Normal University (ECNU). He is CCF Fellow, the vice president of Shanghai Computer Society. His research interests include database, data management, and data-driven applications such as education technology (EduTech), and logistics technology (LogTech).