中国科学: 信息科学 2022 年 第 52 卷 第 2 期: 358–375
SCIENTIA SINICA Informationis
观点与争鸣

专用处理器比较分析

鄢贵海*, 卢文岩, 李晓维, 孙凝晖

中国科学院计算技术研究所, 计算机体系结构国家重点实验室, 北京 100190
* 通信作者. E-mail: yan@ict.ac.cn

收稿日期: 2021–08–16; 修回日期: 2021–10–12; 接受日期: 2021–12–03; 网络出版日期: 2022–01–12

国家自然科学基金 (批准号: 61872336, 62002340, 62090020)、中国科学院 2020 年度青年创新促进会优秀会员 (批准号: Y201923) 和中国科学院 B 类战略性先导科技专项 (批准号: XDB44030100) 资助

摘要 微处理器是现代信息系统的核心基础设施. 大数据、人工智能、5G 等技术的快速发展催生了数据量的爆发性增长, 随之对数据处理能力的需求也急剧增长. 专用计算技术被广泛认为是后摩尔时代的计算机体系结构演化的重要方向. 专用处理器技术的发展一直伴生着通用处理器的发展, 数字信号处理技术甚至早于传统意义上的通用处理器. 通用处理器技术的发展, 不仅在商业上取得了巨大的成功, 很多关键技术也被专用处理器吸收借鉴用于提升专用计算的性能、优化可编程性等. 本文主要分析了数字信号处理器 (DSP)、图像处理器 (GPU)、深度学习处理器 (AI 芯片) 和网络处理器 (NPU) 的关键技术特征, 并进一步对专用计算架构未来发展可能涉及的关键点作出了简要的评述.

关键词 专用处理器, 数字信号处理, 图像处理, 深度学习, 网络处理

1 引言

如果不考虑成本因素, 一颗理想的处理器应该可以像 CPU (central processing unit) 一样通用, 像 DSP (digital signal processor) 一样处理数字信号, 像 GPU (graphics processing unit) 一样处理图像数据, 像 NPU (network processing unit, 网络处理器, 也简称 NP) 一样处理网络数据包, 像 “矿机” 一样竞争加密货币共识算力, 像神经网络芯片一样运行深度神经网络训练和推理等. 但是, 实现如此万能的处理器芯片是不现实的, 至少从经济成本角度不具备可行性, 专用化就成为了发展的必然 [1]. 专用处理器并不是通用 CPU 完全改弦易辙, 而更像是基于通用处理器技术的一种分化. 所以我们看到现代 DSP、基带处理器、网络处理器的很多成功产品都包含一个甚至多个通用 RISC (reduced instruction set computer) 核来做系统管理、运行操作系统、与主机通信, 将协处理器也变成一个具备自我管理能力的主动设备. 从时间上看, DSP 可能也是出现最早的计算芯片, 在集成电路发明 (1958 年) 之前, 德州仪器 (TI) 公司已经在大批量生产硅晶体管器件. TI 公司在 1967 年发明手持计算器, 1971 年研制了单芯片微型计算机. 在此之前的 “DSP” 只能称为利用分立器件信号处理 (processing), 还不是名副

引用格式: 鄢贵海, 卢文岩, 李晓维, 等. 专用处理器比较分析. 中国科学: 信息科学, 2022, 52: 358–375, doi: 10.1360/SSI-2021-0274
Yan G H, Lu W Y, Li X W, et al. Comparative study of the domain-specific processors (in Chinese). Sci Sin Inform, 2022, 52: 358–375, doi: 10.1360/SSI-2021-0274

其实的 “processor”. DSP 没有被冠名某个 “PU” 的称呼也许正是由于出现过早, 当时 “PU” 的称呼还没流行起来. CPU 最早出现在 1971 年, GPU 出现在 1993 年 (虽然当时的名称还不叫 GPU), 网络处理器 (NPU) 出现在 1999 年. 从这些时间关系上看, 我们大体可以看出人们首先是对信号处理有需求, 然后才扩展到其他更普遍的数据处理需求上, 因此有了对通用 CPU 的需求. 再由于应用的驱动, CPU 难以满足性能要求, 进而发展出了 GPU, NPU 等更专用的计算芯片. 从这个意义上看, 通用 CPU 技术可以视为处理芯片的基本技术, 在此基础上发展了高性能 CPU、现代高效能 DSP [2]、高吞吐 GPU、高通量 NPU [3] 等各种 “XPU”. 个人认为分析这些 XPU 的结构特征有助于更深刻地理解 “通用” 和 “专用” 的本质差异.

针对专用处理器有很多关键问题, 包括: 芯片在架构上有什么差异, 各自具备什么样的软件生态, 能否取得商业上的成功的决定性因素是什么等, 而答案也莫衷一是. 本文的重点是试图从架构层面去看待这些不同类别的专用处理器芯片的差异, 帮助我们预测未来架构的发展趋势, 对于专用处理器技术未来的发展做了些许开放性的讨论, 抛砖引玉. 其次, 本文主要讨论经典专用处理器的演化, 而把通用 CPU 的发展作为背景而暂不加以专门讨论. 同时, 本文主要以 DSP、GPU、AI 芯片和 NPU (网络处理器) 为主要参考对象, 其中 DSP 以 TI 公司的 C6000 系列为主要参考, GPU 以英伟达 (Nvidia) 公司的 Tesla 架构为主要参考 [4], AI 芯片以寒武纪的 DianNao [5] 深度学习处理器和 Google 公司的 TPU (tensor processing unit) [6] 为主要参考, NPU 以迈络思 (Mellanox) 公司 (已经被 Nvidia 公司收购) 的 NP-5 和因特尔 (Intel) 的 IXP [3] 为主要参考, 均为各个公司比较有代表性的产品. 限于篇幅, 本文尚未将新出现 DPU (data processing unit) [7] 芯片架构纳入讨论, 相关内容将作为未来工作.

专用计算架构的设计难度不亚于通用 CPU, 核心目标就是 “有条件地” 高性能. 无论是一个 3 W 的 DSP, 还是一个 300 W 的 GPU, 也无论是面向哪一个专用领域定制化的设计, 都追求在给定的功耗、芯片面积约束下实现高性能. 然而, 这个问题的复杂性在于专用计算并不仅仅是设计几个运算单元, 配合几条数据通路那么简单, 它涉及到 IO 子系统、操作系统内核、网络协议栈、访问安全、虚拟化、二次开发的方便程度等层面的问题, 其中任何一个层面的问题的专业性都极强, 要能融汇贯通并能系统地组织起来是一个巨大的挑战. 挂一漏万, 本文的内容只代表笔者的观点.

本文余下内容安排如下: 第 2 节阐述了专用处理器的基本概念, 第 3~6 节分别介绍了数字信号处理器 (DSP)、图形处理器 (GPU)、AI 芯片和网络处理器 (NPU) 4 类重要的专用处理器的基本特征, 第 7 节表述了笔者对于专用处理器的几点思考, 据此在第 8 节提出了构建专用处理器系统结构的关键点, 第 9 节总结全文.

2 专用处理器的基本概念

专用处理器 (或专用加速器), 顾名思义, 就是用于处理 “特定应用” 的处理器, 相对于通用处理器而言, 这类处理器性能更高、功耗更低、通常价格也更便宜, 但是使用范围也相对有限. 计算芯片产业在过去 50 年的发展历程中, 比较成功的专用处理器门类只有数字信号处理器 (DSP)、图形处理器 (GPU) 和网络处理器 (NPU), 这是 20 世纪 90 年代就已经基本定型的格局. 在过去 5 年中, 用于处理深度学习的神经网络处理器 (AI 芯片) 也开始快速发展, 比较成功的案例包括 Google 公司的张量处理器 TPU [6]、寒武纪公司的 DianNao 系列深度学习处理器 [5] 等. 专用处理器的最终目标不是替代通用 CPU, 而是与现有的通用 CPU 技术协作, 即将部分 CPU 运行效率低下的应用卸载 (offloading) 到专用加速器上运行, 通过构建异构计算平台来高效地处理计算任务. 从产业生态的视角来看, 相比于通用处理器的硬件与软件分离的 “水平” 模式, 专用加速器更注重软硬协同的 “垂直” 发展模式.

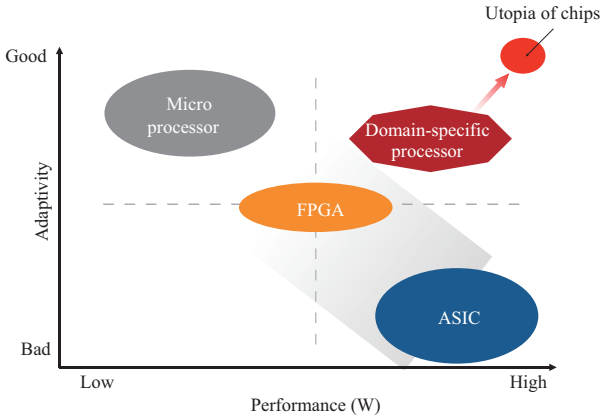

图 1 (网络版彩图) 不同类型芯片效能和适应性相对比较示意

Figure 1 (Color online) Efficiency and adaptability of different chips

图 1 从芯片的效能和适应性两个维度来刻画芯片的特征. 这里效能指的是单位功耗下提供的计算能力, 适应性就是通常意义下的通用性. 业界通常将数据处理芯片大体分为三大类: 处理器芯片、ASIC (application specific integrated circuit) 芯片和 FPGA (field programmable gate array) 芯片. 处理器芯片包括 CPU, GPU, DSP 等, 是用户可编程的芯片; ASIC 是面向特定应用 (application-specific) 的专用集成电路 [8], 通常也称之为全定制芯片, 不可编程; FPGA 器件属于专用集成电路中的一种半定制电路, 是可 “编程” 的逻辑列阵, 利用查找表来实现组合逻辑, 但 FPGA 的 “编程” 与处理器芯片的软件编程不同, 主要是配置逻辑, 可以理解为硬件编程. 从相对性能来看, ASIC 芯片最好, 处理器芯片最差, FPGA 介于二者之间; 但是从应用的适应性来看, 处理器芯片最好, FPGA 次之, ASIC 芯片最差. 值得注意的是这种分类标准并不是按照电路制造工艺, 例如处理器芯片和 ASIC 芯片本质上都是全定制的集成电路, 处理器芯片本质也是一种 ASIC, 但与通常意义上 ASIC 的最大差别还在于是否具有指令集, 有指令集的就更类似传统的处理器, 反之就归类为 ASIC. 此外, 处理器芯片由于其使用广泛、出货量大, 与软件生态联系尤其紧密, 所以将其独立为一个大的类别.

专用处理器芯片的研发追求达到效能和适应性的一个新的帕累托最优 (Pareto optimality): 效能接近 ASIC, 但是适应性向处理器芯片靠近. 在效能上, 专用加速器通过定制化实现远高于通用处理器芯片的效能; 在适应性上, 从面向特定应用 (application-specific) 的 ASIC 范式进化为面向特定领域 (domain-specific) 的新范式, 不妨称之为 “DSIC (domain-specific integrated cirucuit)”. DSIC 与处理器芯片相比虽然弱化了通用性, 但与 ASIC 相比也强化了适应性.

无论是 DSP、GPU、AI 芯片、NPU, 还是现在更新的各种 “XPU”, 都是处理数据的芯片, 最终都需要执行二进制代码的程序来完成计算. 因此专用处理器设计也大都需要涉及如下 6 方面内容: (1) 约定二进制代码的格式, 即指令; (2) 需要将指令变换为机器码, 即汇编; (3) 为了提高编程方便程度, 需要将高层程序语言转换为汇编语言, 即编译; (4) 为了提高编程的效率, 提供了各种编程环境, 即集成开发环境 (integrated development environment, IDE); (5) 充分复用高度优化的代码, 即应用程序库; (6) 为了方便程序调试, 还需要提供各种仿真工具, 即仿真器 (emulator). 所以, 从系统抽象层次来看, 与通用处理器几乎没有区别. 但是不同的 DSIC 侧重点不同, 有些 DSIC 只提供 API (application programming interface) 方式的调用, 例如早期的 GPU, 将编译、汇编等过程全都凝结在运行时库中, 从用户角度看, 调用过程与使用 OpenCL [9] 中的 “内建核函数 (built-in kernels)” 类似, 与调用普通的

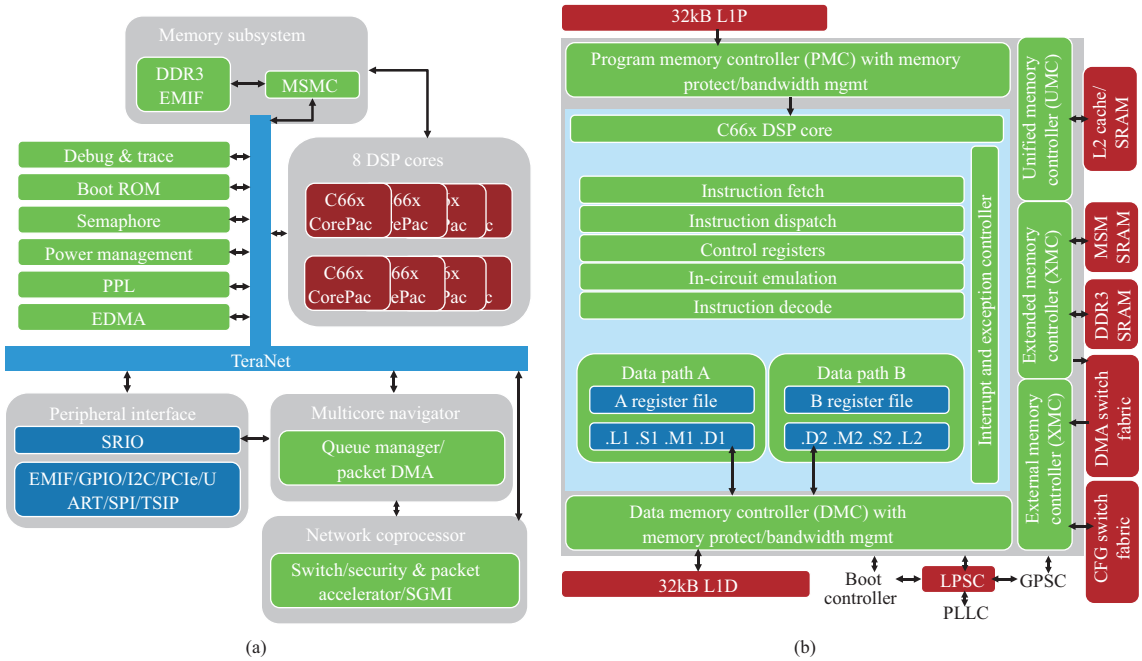

图 2　(网络版彩图) TMS320C6678 结构图

Figure 2　(Color online) Computer architecture of TMS320C6678. (a) Top block diagram; (b) C66x block diagram

库函数过程相同; 虽弱化的可编程性, 但是强化了用户使用的便利性. 但也有些 DSIC, 如 DSP, 使用了大量底层编程, 虽编程难度高, 但方便精确地性能调优.

3　DSP: 灵活的数据格式

DSP 也许是最早出现的专用集成电路. DSP 的使用范围非常广, 从简单的 MP3 播放器到最新一代的 5G 通信都有使用场景. 常见的 DSP 大多带有丰富的外设接口, 例如 PCIe、以太网、UART、I2C 等, 尤其在很多嵌入式设备中, 丰富的外设接口对于提高系统的集成度、降低成本和功耗都有很大帮助, 所以很多 DSP 产品也演变成带有丰富外设接口的 SoC (system on chip) 芯片, 如图 2(a) 所示. 但是 DSP 最大的特点还是进行数字信号处理的核. 大多数 DSP 由于使用场景多为移动设备, 或者只是作为 CPU 系统的数据输入前端, 在系统中的地位并不高, 通常在功耗、散热等方面都不可能给予太高容限, 所以功耗敏感、计算位宽对 DSP 很重要, 定点、浮点, 半精度、单精度、双精度, 16 位、24 位、32 位、40 位等各种数据格式规范 “五花八门”. 在寻址上, DSP 对于数据对齐方式也最灵活, 设置了大量专门的指令对数据进行对齐操作.

TI 公司是 DSP 芯片的龙头, 被媒体评为是半导体行业利润率最高的公司. 2019 财年营业总收入 144 亿美金, 税后净利润高达 50 亿美金, 利润率高达 35%. 作为比较, 同期 Intel 收入 720 亿美金, 利润率 29%; 英伟达总营收 110 亿美金, 利润率 25%. TI 公司的 DSP 主要分为 3 大系列: C2000 系列, 集成了 AD 转换、Flash 存储等, 主要用于控制马达、变频器等工控产品; C5000 系列, 16 位定点, 主要用于便携声音、视频、机顶盒等设备; C6000 系列, 采用了 VLIW (very long instruction word) 架构, 每秒执行指令峰值可达百亿条, 主要用于数字通信、图像增强、传输、加密解密等对性能要求更高的场景. 下面就以比较复杂的 C6678 为例做简要介绍, 其顶层架构如图 2 所示.

粗略观察 DSP 核其实与通常的 RISC 核没有太多区别, 如图 2(b) 所示: 都包括了取指令、指令分发、译码、寄存器读写、Load/Store、计算执行等环节, 但微体系结构有非常显著的特色. 例如, 普通采用超长指令字 (VLIW) 架构、突出的浮点处理能力、指令与数据分离等, 分析如下.

3.1 通过 VLIW 架构提高性能

在 C6000 系列的 DSP 中, 采用了超长指令字 (VLIW) 技术, 性能的提升主要是通过引入 SIMD (single instruction multiple data) 来实现. 从 2 路 16 位、4 路 8 位 SIMD 操作, 到 8 路 16 位、4 路 32 位向量操作. 为了支持较宽的向量化操作, C66x 系列 DSP 设置了 8 个功能单元、两组寄存器堆文件、两条独立数据通路; 每组寄存器文件包含 32 个 32 位通用寄存器, 而且可以支持 8, 16, 32, 40, 64 位等非常灵活的数据位宽打包存储. 例如一个完整 32 位寄存器连同相邻寄存器的低 8 位存储一个 40 位的浮点数, 同时相邻寄存器的高 24 位还可以用于存其他的数. 乘法器支持 128, 40, 64 位数据. 显然支持那么多 “非标” 的定点和浮点数, 如何来安排寄存器的分配成为一个很有挑战的问题.

3.2 浮点乘性能突出

功能单元的乘法计算能力极强, 8 个功能单元的每一个都能做到单个周期完成一条指令的执行, 包括乘法 (.M). 单精度浮点乘在 1 个周期内完成, 双精度浮点乘在 4 个周期内完成, 而且可以支持不同精度的浮点数直接相乘. 算术 (.S)、逻辑 (.L) 指令用来处理算术、分支计算、逻辑运算, 此外, 逻辑指令还可以在两个周期内完成对复数进行 90°, 270° 旋转操作, 计算复数共轭等操作. 作为比较, 2006 年 Sparc-T1 处理器的浮点单元 (floating point unit, FPU) 执行一个单精度乘需要 7 个时钟周期. 其实, 乘法器的性能对于浮点性能至关重要, 现在 FPGA 芯片为了突出计算性能, 甚至把乘法器硬核的数量作为一个重要指标 (FPGA 中通常将其称之为 DSP 核).

3.3 指令缓存和程序缓存分离

将指令与数据分离也就是著名的 “哈佛结构”, 一级程序缓存 (L1P) 采用直接映射, 一级数据缓存 (L1D) 采用多路组相连. 这样导致了缓存替换策略的不同, L1P 采用新缓存行替换同一位置的旧缓存行, 采用读 – 分配 (read-allocate) 策略. 相较而言, L1D 复杂的多, 采用了最近最少使用 (least recently used, LRU) 替换策略和回写 (writeback) 机制: 当数据被更新时, 并不立即更新相应的缓存位置和存储器地址, 而只做 “dirty” 标记, 只有数据被替换出缓存, 或者手动启动一致性操作指令, 或出现长距离访问 (此时所有高速缓存的局部性都极有可能被破坏), 才会写回到存储器. 这也说明指令的局部性是比数据的局部性显著得多, 而且对于核而言, 指令缓存是只读的, 而数据缓存可读可写, 从这个意义上看, 将二者分开也是有好处的. 此外, DSP 的缓存还支持很多先进的管理功能, 例如强制冻结模式 (freeze mode), 可以防止中断程序破坏已经建立在缓存中的数据局部性, 降低中断恢复后 “冷启动” 性能开销. 这些操作也全都由程序员来完成.

3.4 硬件指令支持一致性管理

多核并不是 CPU 的 “专利”, C6000 系列也提供多核的 DSP, 由于多核引入会导致数据一致性的问题, C66x 系列 DSP 也提供了栅栏指令 (MFENCE) 来处理缓存回写, 强制或阻止一致性操作的执行等, 方便程序员管理数据一致性.

3.5 硬件化的带宽管理防止运行阻塞

DSP 核中还设置了硬件化带宽管理, 负责管理一级数据缓存 (L1D)、一级程序缓存 (L1P)、二级缓存 (L2)、寄存器配置总线等 4 类资源的访问优先级. 访问发起方包括 DSP、外部 DMA (enhanced direct memory access, EDMA)、内部 DMA (internal direct memory access, IDMA)、数据一致性操作. 管理按照每次访问授予优先级, 而不是按照访问类型固定优先级, 通过设置竞争强度计数器来反映对资源的 “饥渴” 程度, 即便是最低优先级的访问, 随着等待时间增加, 优先级就会逐渐升高, 当达到最长等待周期数, 就会强制授予一次访问. 而这些都是硬件管理的, 程序员只能设置最长等待时间, 不能设置竞争计数器. 这样的硬件化维护资源公平性的设置在 CPU 中并不常见.

由以上分析可以看出, DSP 作为一类典型的专用处理器, 其结构与数字信号处理需要丰富的 IO 接口便于集成, 强大的浮点处理能力支持高带宽的信号处理, 还提供了丰富的底层数据通路的控制手段方便专业用户的性能调优.

4 GPU: 数据并行的典型代表

GPU 是专为图形 (graphic) 处理设计的专用处理器. 随着多媒体可视化需求的爆发, 传统的 CPU 是无法应对每秒动辄数百兆的视频渲染等任务. 高清图像、视频数据天然具备数据并行的特征, 可以通过高度的并行性来同时计算像素块中所有像素的色度、亮度等数据. 图 3 显示的是英伟达公司研发的经典 Tesla 架构的 GPU, 之所以经典, 是因为从这一代架构开始, GPU 开始朝着通用 GPU (即 GPGPU) 发展, 为后续 GPU 在深度学习领域的广泛应用奠定了基础.

4.1 采用了单指令多线程 (SIMT) 的结构

单指令多线程 (single instruction multiple thread, SIMT) 是英伟达针对其 GPGPU 提出来的一种处理器执行模型, 与传统的单指令多数据 (SIMD) 略有差别. SIMT 具备了动态调度机制, 每个线程 (thread) 都可以独立处理分支, 而 SIMD 更像 VLIW, 需要软件来管理批量取数 (batch load) 和分支. 其实这点灵活性给 GPU 的优化带来了很多麻烦, 之所以要使每个线程都可以独立分支, 本质是为了更好地服务于通用 GPU (即 GPGPU). 因为在图像处理之外的计算负载很少能找到能同时成千上万个同构的线程, 否者直接采用 SIMD 就满足要求了. 但是为了控制管理这些线程的复杂度, GPU 将这些线程组织成群 (在英伟达公司的 GPU 技术里称之为 Warp, 在 AMD 公司的 GPU 体系中称之为 Wavefront, 若无特殊说明, 后文均以 Warp 为例), 调度是以 Warp 为单位, 这就意味着同一个 Warp 中的线程必须以分支范围为界, 所有分支必须在同步前汇合 (converge), 这就很容易导致其他线程等待而不能充分利用 GPU 的运算资源. 这在运行控制密集型负载时负面影响尤为突出. 为了降低由于同一 Warp 内某些线程分叉 (diverge) 带来的同步开销, 在 Tesla 架构下, 每个 Warp 的大小被限制在 32 线程, 只有上代产品的一半.

4.2 GPU 对于 AI 的支持

由 AI 以及渲染需求带来的 GPU 架构变化, 以 NVIDIA Ampere 架构和 AMD RDNA/CDNA 为例. 首先, 支持多精度/混合精度训练. 大多数深度学习模型使用 32 位单精度浮点数 (FP32) 进行训练, 而混合精度训练的方法则通过如 16 位浮点数 (FP16) 进行深度学习模型训练, 从而减少了训练深度学习模型所需的内存. 同时由于 FP16 运算比 FP32 运算更快, 进一步提高了硬件效率. AMD 设计 “matrix fused multiply-add” 指令集进行混合精度计算, 支持 8 位整型 (INT8)、16 位半精度浮点 FP

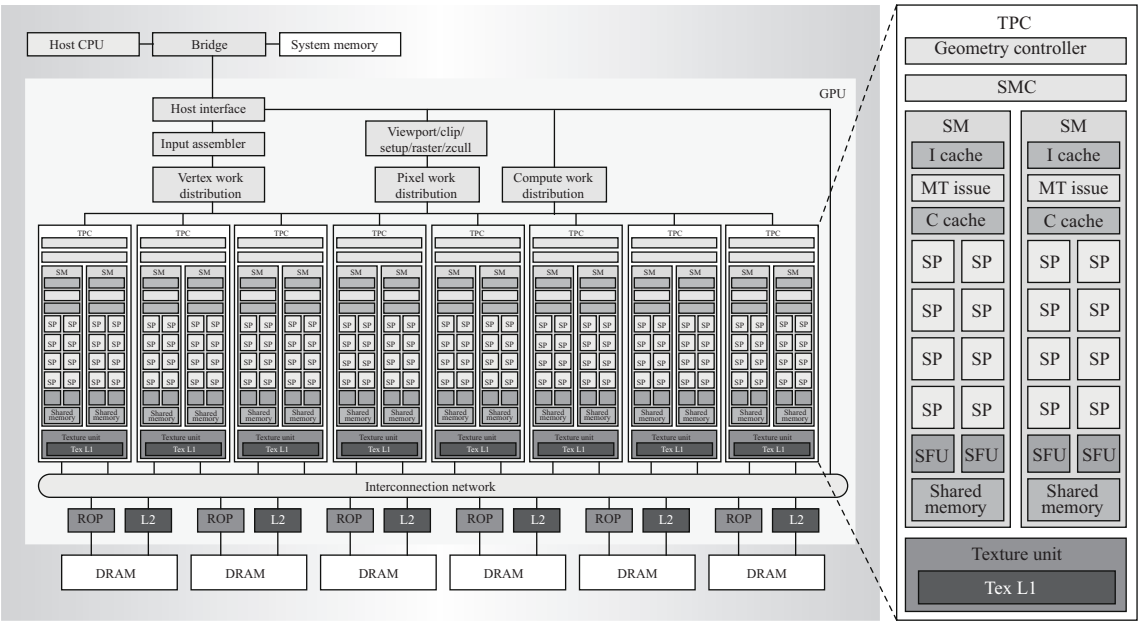

图 3 (网络版彩图) Nvidia Tesla 结构图 [4] @Copyright 2008 IEEE

Figure 3 (Color online) Computer architecture of Nvidia Tesla [4] @Copyright 2008 IEEE

(FP16)、16 位 "brain" FP (BF16) 和 32 位单精度浮点 (FP32). NVIDIA Ampere 架构进一步提升了 FP32 的计算速度, GeForce RTX 3090 实现 35TFLOPS (采用 FP32), 是其上一代 Turing 的两倍. 其次, 引入新的计算资源. NVIDIA Ampere 架构包括: 可编程的 shading cores (由 NVIDIA CUDA core 组成), 新一代 ray tracing core (RT core) 用于加速光线追踪时的 bounding volume hierarchy (BVH) 遍历, 以及场景划分. 其中 tensor cores 用于提升神经网络的计算和推理性能. AMD 的 RDNA 架构最主要的核心改动就是将原先的矢量集架构改成了标量集架构. 之前 AMD 的 Wavefront 是 64 个线程, 这一代的架构改成了 32 个线程. 这一点与 Tesla 架构采用 32 线程的 Warp 设置是一致的. 采用 32 个线程的配置的优点是可以减少 Warp 内分支的性能开销, 单个 Warp 也只需要更少的寄存器, 同等的硬件资源可以支持更多的 Warp (wavefront), 从而有效地提高吞吐量和隐藏延迟.

4.3 现场调度做到 "零开销 (zero-overhead)"

传统意义上线程的调度涉及保护现场、上下文切换、指令重新载入等环节, 通常来看线程的调度会破坏缓存内数据的局部性, 导致缓存缺失率的上升, 而且现场状态保存等也需要消耗时间, 所以理论上看是不可能零开销的. 但是 GPU 中线程由于量级很轻, 同一个 Warp 内的线程调度并不需要换入、换出现场的状态 (主要是寄存器内容), 所以可以做到近似零开销. 在 Tesla 架构下, Warp 是调度的最小单位, 指令的调度也就是 Warp 的调度, 在 Warp 间调度开销也很低. 每个 Warp 由 32 个执行同一指令的线程组成, 每一个线程 (thread) 都有自己的寄存器状态, 独立判断分支的状态执行, 由于分支决策依赖于数据, 所以同一个 Warp 中的不同的线程可能会 "分道扬镳" (diverge), 同一个 Warp 中的线程通过 Barrier 来同步, 只需要执行一条流式多处理器 (streaming multiprocessor) 指令. 每一条线程从创建、调度、同步的效率都很高. 具体的调度机制采用记分板机制来实现, 调度的优先级涉及到 Warp 类型、指令类型和公平性, 由于涉及到技术机密, Nvidia 公司没有公开更细节的内容. 类似的降低指令调度开销的技术在网络处理器 (NPU) 里也常见到. 调度做到 "零开销" 对于优化公平性很

有帮助, 不会因为计算资源轮询共享而牺牲性能.

4.4 引入特殊超越函数处理单元

GPU 中的特殊功能单元 (special-function unit, SFU) 用于计算超越函数, 例如倒数、平方根、幂函数、对数、三角函数等. 基本方法是采用查表结合二次插值的方法来计算, 做到了性能、误差和面积的平衡. 性能上做到了每周期出一个结果, 误差上界可控, 面积远小于单纯的查表方案. GPU 中的 SFU 和现在流行的神经网络处理器里用到的 "Sigmoid" 函数有类似的地方, SFU 中的计算方法用来处理 Sigmoid 计算是足够用的. 此外, 之所以现在 GPU 被广泛应用于神经网络计算并不单纯因为性能高, 而是 GPU 中的流式处理器 (streaming multiprocessor) 的结构与神经网络的矩阵 – 向量 – Sigmoid 的操作比较匹配.

4.5 隔离难以通用的部分

GPU 中的最后一个环节光栅操作处理器 (raster operation unit, ROP), 是面向图像处理专用化程度最高的一部分, 在其他通用计算中应该很难利用起来, 这也许是在 Tesla 架构中, ROP 作为独立单元与纹理处理器 (texture processor cluster, TPC) 解耦合的原因之一. ROP 通过片上网络与 TPC 交互, 而且作为离最终图像格式最近的操作, 每个 ROP 与 DRAM 的一个分区直接成对连接, 实现近内存 (near-memory) 的结构, 每个 DRAM 分区有 16 GB/s 的带宽独立访问. 从最终的版图来看, 虽然 ROP 的数量 (6 个) 远远少于 TPC 中 SM 的数量 (16 个), 但是其占有的面积 (连同其 L2 缓存) 还是与 SM 相当的.

GPU 技术的主要趋势是从固定图形函数流水线向可编程并行处理器发展. 人工智能的这一波浪潮帮助 GPU 扩大了 "群众基础", 加之英伟达之前不遗余力地对 CUDA (compute unified device architecture) 推广, 使得 GPU 往 "通用" 方向做了很多有意义的拓展. 但是今天的 GPU 用于做非图像类应用 (或者通用计算) 普及程度其实远没有想象那么高. 乘着人工智能的东风, 近年来 GPU 经历了 5 年的快速增长, 但全球 GPU 的出货量在 2018 年第 4 季度甚至出现了环比下降 2.7% 的回撤趋势. 从某种意义上说, 尽管云计算市场的快速发展有利于 GPU 专用服务器的增长, 但是除了深度学习催生了上一轮 GPU 的快速发展到达顶峰后, 在其他应用领域, GPU 相对 X86 的通用 CPU 没有显著的替代效应. 而且, 随着 AI 专用芯片技术的逐渐成熟, 对 GPU 在深度学习领域的市场占有率将很有可能呈现一定的替代效应.

5 AI 加速器: 大规模张量处理

随着深度学习算法效果在图像处理中取得突破, 深度学习在越来越多的应用领域被应用. 在过去 5 年中, 用于处理深度学习的神经网络处理器 (AI 芯片) 快速发展, 其中比较成功的案例包括寒武纪的 DianNao 系列深度学习处理器 [5] (图 4)、Google 的张量处理器 TPU [6] (图 5) 等. AI 加速器大多针对机器学习中张量运算展开加速, 多基于 SIMD 方式实现, 单条指令通常可以完成一个矩阵的乘法运算, 因此也称为张量处理器. 除了张量处理之外, 还会伴有一些激活函数处理等非线性操作, 运算量相对较小, 实现起来也相对简单. 此类加速器重在张量处理, 控制相对简单, 通常不会集成通用处理核.

5.1 大规模张量运算阵列

在机器学习中, 张量处理占据整个模型中 80% 以上的运算量, 并且多为规整的向量乘累加运算, 因此几乎每一类 AI 加速器都集成了一个大规模的张量运算阵列来加速张量处理. 阵列中每个运算单

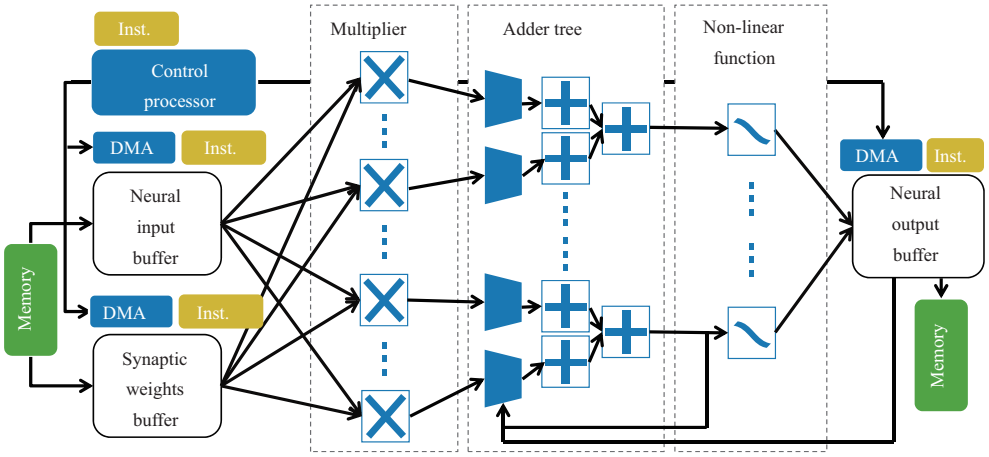

图 4 (网络版彩图) 寒武纪 DianNao 结构图

Figure 4 (Color online) Computer architecture of Cambricon DianNao

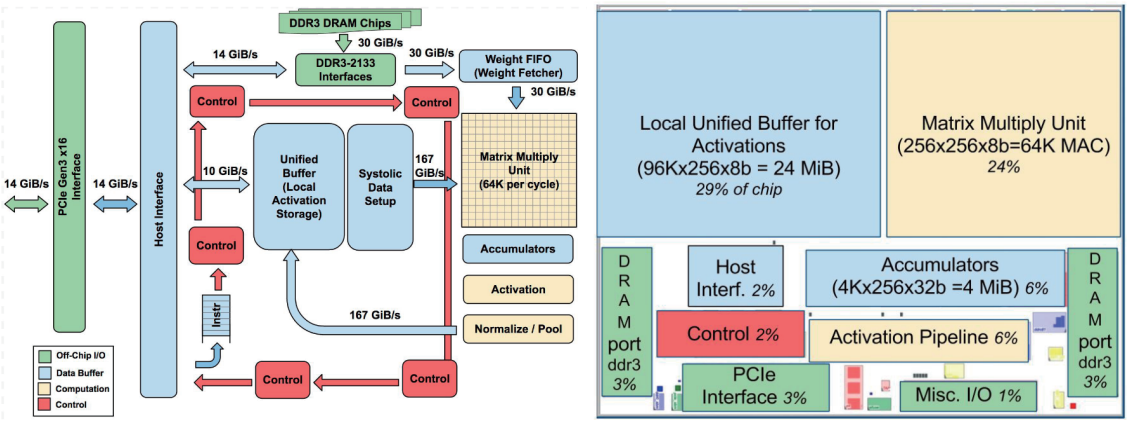

图 5 (网络版彩图) Google TPU 结构图 [6] @Copyright 2017 IEEE

Figure 5 (Color online) Computer architecture of Google TPU [6] @Copyright 2017 IEEE

元多为乘加单元再辅以局部寄存器. 如寒武纪的第 1 代 DianNao 集成了 16×16 规模的 NFU (neural function unit). 寒武纪的 NFU 也是最早期最有代表性的张量处理单元. 之后寒武纪经过多次迭代在 DaDianNao [10] 中集成多个 NFU 可实现更加高效的运算. 随后, Google TPU 推出了基于脉动阵列的张量处理单元 256×256 的 MAC (multiply add components) 阵列, 一个周期可以完成 64k 次乘累加运算. 为了高效地给张量运算阵列提供数据, 并及时回存中间结果, 在片上通常会集成大容量的缓存. 在 DianNao 中集成了专门用于存储神经元和突触的片上存储器, 在 Google TPU 中也集成了 24 MB 的片上存储. 同时, 为了尽可能高地提高细粒度并行, 可以灵活地将计算阵列配置成输入神经元共享、特征图共享或者输出神经元共享的方式 [11], 最大化片上数据的复用性, 减少由于访存带来的性能开销.

5.2 可变精度处理

相对于 DSP 等加速器, AI 加速器对数据精度要求较低, 多基于定点数实现. 并且为进一步提升计算效率, 每个神经网络模型, 甚至网络的每层数据精度都分别对待. 可变精度处理也是 AI 加速器的一

大特点, 通常 32 位/16 位/8 位/4 位/2 位/1 位定点数都有可能. 无论张量运算单元, 还是存储单元都有可变精度的相应设计. 寒武纪 DianNao 专门探讨了数据位宽对精度的影响, 最早基于定点数实现了加速器. 在 Google TPU 中, 16 位计算结果存储在由 32 位单元组成的 4 MB 累加器 (4k×256×32 bits) 中. 存储管理单元 MMU (memory management unit) 每个周期产生一次含 256 个单元的部分和. MMU 存储一个 64 kB 的权重块 (tile). 当使用 8 位权重和 16 位激活函数操作时, MMU 以一半的速度执行. 当两者都是 16 位时, 计算速度为峰值的四分之一. 每周期可以读或写 256 个数值, 并做矩阵乘操作或是卷积操作.

5.3 稀疏处理

除了可变精度之外, 在神经网络模型中权重数值和神经元数值中多存在一些无效零值, 再加之模型侧剪枝操作进一步产生更多的零值. 零值的引入给 AI 加速器设计带来了新的优化维度: 跳过无效运算进一步提升效率. 针对无效运算的处理通常有两种方式: (1) 运算之前去除掉无效值; (2) 运算时去除无效值. 针对第 1 种实现方式, 典型代表如寒武纪的 Cambricon-X 加速器, 需要对数据管理单元 MMU 做出相应的改进, 对运算单元改动较小. 针对第 2 种实现方式, 典型代表如 FlexFlow-Pro [12], 需要对运算阵列做出一定调整.

AI 加速器是典型专用处理器, 它弱化了指令级可编程性, 但是强化了用户使用的便利性, 没有统一的指令集, 各厂家指令实现方式各不相同. 如寒武纪偏向于以 RISC 方式定义自有的指令集, 而 Google TPU 指令则依照 CISC (complex instruction set computer) 的设计思路, 平均 CPI 在 10∼20 之间. 对于 AI 加速器更加注重的是直接与各种开源的开发框架对接, 如 TensorFlow, MXnet, Caffe, Pytorch 等, 用户直接对接开源框架, 而无需关注具体指令实现. 为此寒武纪也实现了自己的完整的软件工具链, 方便用户部署应用.

6 NPU: 为网络数据包处理而生

计算机网络是计算机系统发展过程中的一个伟大发明, 网络是大规模并行计算、分布式计算的必要条件. IBM 于 1974 年发布 SNA (system network architecture) 系列网络协议, 主要解决 IBM 的大型机与外围节点的通信问题. 这些节点并不是完整的计算机, 而是像用于 ASCII 图像显示终端的 IBM-3174 控制器、打印机等设备. 1974 年之前程序还写在纸带上, 主要处理模式还是批处理; SNA 引入以后, 开启了事务处理的先河, 把网络通信中容错任务交给了网络协议来处理, 并且基于 SNA 后来发展出了应用程序接口 (API) 等概念. 随着后续几年越来越多的设备开始采用网络来连接, 不同厂商提供了不同的网络, 为了解决不同网络间的互连互通 (internetwork communication), 美国标准化组织于 1981 年提出了经典了开放系统互连 (open systems interconnection, OSI) 7 层模型 [13], 这个参考模型一直沿用至今, 仍未过时.

网络处理器的出现是网络技术发展的必然. 随着 OSI 模型的普及和广泛接受, 在 2000 年前后, NPU 还是学术界研究的热点领域. 第 1 颗网络处理器于 1999 年问世, 随后得到了许多半导体公司、网络设备厂商的关注, 据不完全统计, 前后有 30 余家厂商完成了 500 余款 NPU 的设计, 和现在的各种 “XPU” 的多样性相比有过之而无不及. IBM、因特尔、思科、EZChip (于 2015 年被 Mellanox 收购) 都推出了相应的系列产品, 典型如 Intel 的 IXP 系列和 Mellanox 的 NP 系列网络处理器.

各家 NPU 产品虽然各有差异, 覆盖不同的协议层次, 面向不同的协议内容, 但是它们的结构模块具有相似性, 例如都包含的模块包括通用处理器核、队列管理单元、路由管理、缓存管理、IO 接口管

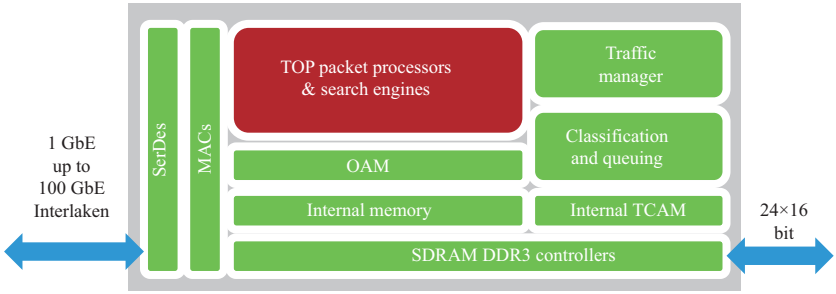

图 6 (网络版彩图) Mellanox NP-5 网络处理器

Figure 6 (Color online) NPU: Mellanox NP-5

理等. 值得一提的是, 最近兴起的 DPU 架构, 有很多特征借鉴了 NPU 技术, 特别是在对路由、交换、数据包处理的高效支持上, 相关讨论将作为未来工作开展.

6.1 控制平面与数据平面相分离

网络处理器的功能可以从控制平面和数据平面来区分. 判断一个任务属于控制平面还是数据平面的一个简单的方法就是看数据包的路径: 如果一个数据包需要穿过处理器, 就属于数据平面, 如果是由处理器发起, 或者终结在某个处理器, 就属于控制平面. 其中数据平面的计算主要由专用计算引擎完成, 而控制平面的计算主要由通用处理器核完成. 不同于 GPU, 浮点处理性能不是网络处理器的重点, 可以灵活支持 OSI 7 层协议中不同层次的各种网络协议的高效卸载才是网络处理器比拼的焦点.

图 6 为 EZchip 最新推出的 NP-5 网络处理器的架构图, 该架构使用专门的处理器进行网络处理所需的不同任务. NP-5 的上一代 NP-4 是业界首个 100 Gbps 电信级以太网网络处理器, 到了 NP-5 已经把标称性能提高到了 240 Gbps 的高度. 任务优化处理器 (task optimized processor, TOP) 是 NP 系列专用处理器的核心, 是以流水线方式排列的超标量处理器. NP-1 声称专为处理第 2~7 层数据包而设计处理速度为 10 Gbps. 有一个单独用于控制处理器来处理控制平面操作的接口. EZchip 采用微码编程, 具有用于 NP-1 的软件开发环境, 包括汇编器、调试器和仿真器, 此外还提供常用的交换、路由、分片、组包、组播、QoS 等应用程序库.

6.2 面向数据包处理构造的异构核

EZchip 的任务优化处理器 (TOP) 是其最核心的部分, 厂商没有公开太多设计的细节. 根据用户手册得知, TOP 的基本架构是以流水线的方式排列, 处理 4 种类型的任务: (1) 识别和提取各种数据包头和协议; (2) 执行不同级别的查找 (第 2~7 层); (3) 将数据包分配到适当的队列和/或端口; (4) 修改包内容 (例如根据路由表修改链路层目标地址等). 这 4 种类型的任务分别对应了 4 类处理器核: TOPparse, 用于解析得到包头、地址、端口、协议类型等; TOPsearch, 用于根据解析得到的信息查表得到相关的路由、包类别、处理策略等; TOPresolve, 用于处理转发、QoS 控制、更新路由表状态等; TOPmodify, 用于修改、增加数据包内容. 每一个数据包处理都会经历这几个阶段 (stage). 处于同一个 stage 的多个核运行相同的程序, 并同时处理多个数据包. 每一个 TOP 都有独立的程序存储器, 但是并不需要与其他 TOP 核同步. 我们不妨把这种模式称之为 SPMP (single program multiple packets).

TOP 不同类别的核都是通过微码来编程, EZChip 提供汇编器来生成二进制代码. 对于 TOPparse, TOPresolve, TOPmodify 都是 4 级流水: 取指 (fetch)、译码 (decode)、读源操作数 (exe1)、写目的操作数 (exe2). 但是, 每一类核的结构都各不相同, 每一类核都有特殊的指令. 尤其 TOPsearch, 其结构更

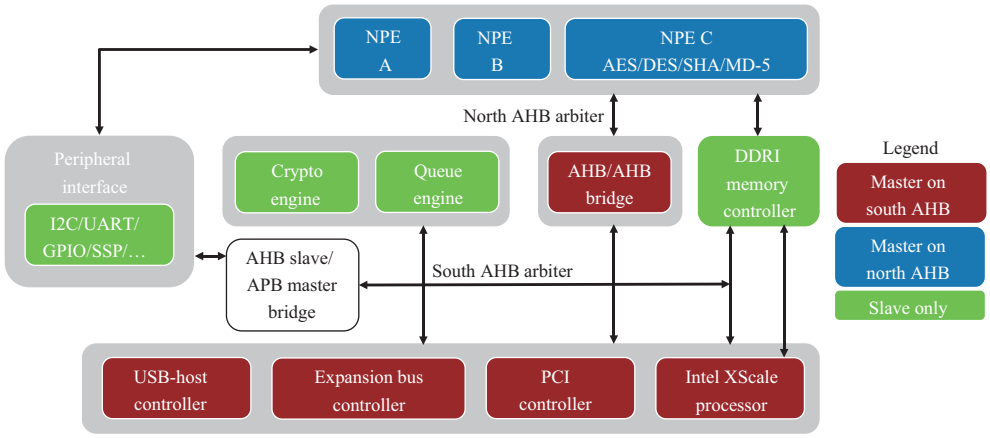

图 7 (网络版彩图) Intel IXP 网络处理器结构图

Figure 7 (Color online) Intel IXP network processor architecture

像是一个内存数据库, 其核心是一个巨大的查找表引擎 (lookup engine).

Intel 的 IXP 系列网络处理器也是很有代表性的架构, 如图 7. 和其他 NPU 一样, IXP 系列处理器也有一个 RISC 处理器核 (32 位的 XScale 核) 作为主控, 用于处理控制平面的操作. 还有一个网络专用的部分, 称之为 NPE (network processor engine), 主要用于处理 CRC 校验、分包/组包、加密解密、哈希计算等操作. IXP45X 系列处理器最多支持 3 个 NPE, 分别为 NPE-A, NPE-B 和 NPE-C, 每个 NPE 都有独立的指令存储器, 但是 3 个 NPE 功能各不相同, 这与 EZChip 的 NP 系列处理器里的多个不同类型的 TOP 处理核类似. 支持带优先级的硬件多线程, 上下文切换可以做到零开销 (严格意义上是一个时钟周期), 这也与 NP 系列处理器是类似的. 所有的 NPE 功能需要采用 Intel 提供的软件, 用户并不需要对 NPE 来单独编程, 只需要按照功能调用即可.

6.3 高度复杂灵活的编程

如果不是因为网络处理器的使用场景完全是面向专业网络开发人员, 奇高的编程难度估计很难像 CPU 编程一样普及了. 不仅仅是 EZchip 的 NPU 是这样, 已经停产的 NXP 的 C5 网络处理器也包括了 5 个结构和功能各异的协处理器和 16 个包含了 RISC 核的通道处理器, 其编程的难度也就可想而知了. 2003 年 PLDI (ACM Conference on Programming Language Design and Implementation, 是计算机编程语言领域的旗舰学术会议) 会议上有学者写了一篇题目颇为形象的论文 “驯服 IXP 网络处理器 (Taming the IXP network processor)” [3] 很能说明当时网络处理器编程困难的问题的. 其实从编程难度来看, 更具异构特征的网络处理器的难度比规则结构的 GPU 更具挑战.

虽然目前网络处理器似乎已经不是研究的热点了, 今天的研究人员听到 NPU 的第一联想可能是更时髦的 “neural” 处理, 而不是 “network”. 甚至体系结构方向研究生选题也很少再以网络处理器相关方向作为研究方向, 而更多的是 AI, 以致于神经网络处理器将曾经风光的 NPU 的名字都 “据为己有”, 有些江河日下的观感. 但个人觉得, 网络作为计算基础设施的 4 大要素之一, 仍然有强大的生命力, 无论是学术研究还是工程实现, 都还有很多问题没有解决. 网络处理器体系结构仍然是一个具有超高技术门槛的处理器类型.

表 1 对比 DSP, GPU, AI 芯片和 NPU

Table 1 Difference among DSP, GPU, AI chips and NPU

	DSP	GPU	AI chips	NPU
Application domain	Digital signal processing, such as video and audio acquisition, sensor signal processing, filtering, and signal enhancement	Image rendering, high resolution video processing, deep learning model training, high-performance computing, etc.	Deep learning, security monitoring, object recognition, action recognition, etc.	Network routing switching, store and forward, network security, etc.
Architecture feature	RISC like, VLIW adoption for high performance; high-performance multiplier and multiple peripherals, and RISC processor core for control plane; multiple IO types and different performance	Isomorphic multi-core, simple control path, single instruction multithreading; high IO bandwidth requirement for processing high-definition and high frame rate images	Large scale MAC operation array, high-capacity on-chip cache; support variable precision and separated inferencing and training; sparse matrix-optimized computational performance	Heterogeneous multi-core, task pipeline; general processor core processing control plane, dedicated core processing data plane; match between IO and the network bandwidth
Programming feature	Programming at assembly language level, combined with part of C language, can control a large number of microstructure level operations, and the performance is highly dependent on programming optimization	Directly call OpenGL, DirectX and other APIs for image processing, and use CUDA extended C language for general programming	Using existing AI programming frameworks to define computation flow diagrams, such as TensorFlow, MXNet, PyTorch, and PaddlePaddle, programming is relatively easy	Microcode programming, similar to DSP, has prominent heterogeneity and complex programming
Performance characterization	Fixed point and floating point have corresponding products, which are determined according to the use scenario	Mainly focus on floating-point performance, and the latest GPU has mixed precision	Variable precision, low precision fixed-point for inferencing; Increased precision and processing FPS when training for image recognition	Highlight fixed-point performance, packet processing delay, core throughput and high real-time requirements

7 关于专用处理器的几点思考

表 1 对前述的 DSP, GPU, AI 芯片和 NPU 从架构特征、编程特点和性能刻画 3 个维度进行了比较, 概述了这 4 类专用处理器的主要区别. 分析可以看出, 专用处理器的设计相比于通用 CPU 而言, 侧重点非常不同. 比如, 不太强调指令集, 也没过多的突出应用生态, 但是非常注重极致的性能优化, 注重与具体计算模式的匹配. 以下是笔者关于专用处理器的本质特征的几点思考, 抛砖引玉.

7.1 专用处理器的指令集架构没有最好的, 只有最合适的

指令级体系结构 (instruction set architecture, ISA) 特征, 通常分为 CISC, RISC, VLIW 三个大类, 前两类大家都比较熟悉, 在此略过. VLIW 一直未在通用市场上取得成功, 在 20 世纪 80 年代, 最有名的一家希望在通用市场推广 VLIW 的公司应该是 Multiflow, 由耶鲁大学 (Yale University) 的计算机

科学家 Josh Fisher 创立于 1984 年, 他们在 VLIW 的编译优化方面做了很多研究, 这家公司只生存了 6 年, 总共只卖了一百多台 VLIW 机器, 随后 Fisher 教授加入了惠普 (Hewlett-Packard) 公司. 虽然公司没有成功, 但对 VLIW 的贡献为其赢得了计算机体系结构领域最高奖 Eckert-Mauchly Award. 20 世纪 90 年代正值摩尔定律飞速发展的时代, Multiflow 研发的 VLIW 处理器的性能一直跟不上 RISC 和 CISC 的微处理器, 最后被淘汰出局. 即便是 Intel 和惠普提出安腾 (Itanium) 架构, 即 IA-64 系列, 又发起了 VLIW 的尝试. 第 1 颗 IA-64 架构的处理器发布于 1999 年, 虽然与 X86 师出同门, 但仍未动摇到 RISC 的 ARM 和 CISC 的 X86 的市场地位, 2017 年安腾彻底退出历史舞台.

VLIW 处理器优势使得控制结构大幅消减, 提升了单位面积的有效计算的密度, 功耗效率也得到了提升, 但与 CISC 和 RISC 相比最大的差异是编译依赖. VLIW 的性能对编译优化极其敏感, 将开发指令集并行 (instruction level parallelism, ILP) 的责任交由编译器来完成, 不仅导致编译器的复杂度大大提高, 而且还带来了一系列兼容性的问题: VLIW 的代码都有很强的机器依赖 (machine dependence), 更新代际的机器执行之前的代码并不会获得性能的明显提升. 此外, 理论性能与实际性能的落差过大始终是 VLIW 绕不过去的坎, 这也许是 VLIW 未获得广泛市场认可的重要原因. 然而, 如果把 VLIW 的目标范围缩小到仅仅是数字信号处理的范畴, 对编译优化的要求也会随之降低, 这就有可能在一个可接受的编译优化复杂度下, 比较充分地利用 VLIW 架构的计算资源. TI 的 C6000 系列 DSP 已经用事实证明了这一技术路线的可行性. 所以, 脱离了实际的应用范畴谈体系结构的优劣都很容易 “刻舟求剑”. VLIW 没有成功代替 X86, 并不能说明 VLIW 就是失败的; X86 服务器出货量占到了所有通用服务器出货量的 90% 以上, 也不能证伪其他技术存在的合理性.

7.2 专用处理器的争夺在 “数据平面”, 而不在 “控制平面”

在软件定义网络 (software defined network, SDN) 技术的发展过程中发展出了控制平面 (control plane) 和数据平面 (data plane), 控制平面是为数据平面准备必要信息, 例如更新路由表、ARP、IGMP、设备管理等, 大多属于非实时性操作; 而数据平面突出性能, 实时响应, 能处理突发性包转发. 至于什么类型的任务应该划分为控制平面, 什么任务划分为数据平面并不是绝对的, 需要对系统的资源有整体的分配. 整体而言, 控制平面突出安全性、可用性, 而数据平面突出性能.

笔者认为把控制平面和数据平面的概念延伸到其他领域也很有借鉴意义. 专用处理器最重要的指标就是性能, 这与数据平面的诉求是一致的; 更重要的是数据平面和控制平面的需求差异必然导致计算架构的差异, 而试图融合二者的差异必将带来效率的损失. 这在 GPU 向 GPGPU 拓展过程中的阻碍可见一斑. 现在比较常见的 ISA, 包括 X86, ARM, 甚至 RISC-V 的领地争夺甚是激烈, 都是控制平面的存量竞争; 而数据平面的竞争更多是开疆拓土, 瞄准的是未来新应用需求, 属于增量竞争.

7.3 指令集对于专用处理器而言并不是形成生态的主要原因

指令集的重要性毋庸置疑, 但专用处理器的指令集更重要的意义与通用 CPU 不同. 专用处理器的指令集更重要的意义在于帮助完成软硬协同的研发迭代, 减少软硬件的沟通成本, 而通用 CPU 的指令集则更多的需要承载生态, 构造壁垒. 本质差异的原因是由 “专用 – 垂直” 的发展模式与 “通用 – 水平” 的发展模式决定的. 例如 Mellanox 的 NP 系列 NPU, 基本没有公开其指令集, 手册上也未公开其内部的通用处理器核是什么型号; IXP 的网络处理器的控制核采用了 XScale 架构, 采用 StrongARM 指令集, 但这部分在整个专用处理器中只占很小比例. 甚至 Nvidia 也并不强调其指令集, 而是强调 CUDA 编程环境, 突出支持 OpenGL, DirectX 的 API.

7.4 芯片的可编程性不是商业成功的决定性因素

决定专用处理器芯片市场地位的是其背后的应用承载的价值大小, 而不是可编程性. 有人把芯片的可编程性当成芯片能否普及的重要指标, 认为可编程性不好, 就意味着芯片不可能取得市场成功, 判断的逻辑很简单: 编程性不好 = 不好用 = 用的人少 = 市场小 = 失败. 其实, 无论是 DSP 还是 NPU, 包括有 CUDA 加持的 GPU, 其编程都不可谓不难, 但是并不妨碍这些专用处理芯片每年数千万颗、数百亿美金的市场容量. 所以决定专用处理器芯片市场地位的还是其背后的应用承载的价值大小. 在数字时代, 有信号的地方就需要 DSP, 奠定了 DSP 的市场地位, 网络时代为 NPU 的规模化应用提供了历史机遇. 性能、功耗、性价比是关键, 而编程是 “专业” 人士的问题, 市场的规模就相对客观, 因为需求是客观存在的, 编程的难易程度不会直接影响市场需求的规模. 类似情况对于 GPU 也是一样.

7.5 专用处理器的性能没有可迁移性

现在看到一些专用处理器芯片的性能都是以 “T” 为单位 —— 每秒万亿次计算! 单片! 最新的 DSP 每秒能执行百亿条 VLIW 指令, 假设每条指令编码 10 个计算操作, 也只能提供千亿次计算. 离 “T” 还差一个数量级. 最新推出的 Ampere 架构 GPU [14], 单精度张量浮点 TF32 (不同于单精度浮点 FP32, TF32 只用 19 位表达: 指数部分 8 位, 与 FP32 相同, 但尾数部分只有 10 位) 训练性能高达 156TFLOPS, INT8 推理性能达到惊人的 624TOPS! 这样的单芯片性能让最新的高性能 X86 通用 CPU 都望尘莫及. AMD 目前最高端的 64 核 “霄龙 EPYC”, 主频 2.6 GHz, 假设单核 IPC (instruction per cycle) 能达到峰值 4, 理论上性能估算也就在 600GOPS 左右, 远远小于动辄数百 TOPS 的量级. 因此, 虽然 GPU 的性能高出 CPU 许多倍, 但是 GPU 并不能将 CPU 上运行的负载都加速几个数量级, 甚至连是否可以运行都是问题. 这不是工程复杂度的问题, 而是可行性的问题. 好比轮船的装载量远远大于卡车, 但是如果要从上海运货到西安, 选择轮船并不可行.

8 讨论: 构建专用处理器系统结构的几个关键点

专用处理器与通用处理器是处理器发展的两个互补的方向, 虽然单独一类专用处理器的市场要远小于通用处理器, 但是多类专用处理器的市场总和将远远大于通用处理器的市场. 而且, 专用处理器的发展将会在很多增量的应用市场中占有绝对的性能优势, 且受到通用计算的生态限制更少, 有利于专用架构逐步扩展去覆盖更长尾端的应用. 在过去处理器芯片发展的 60 年里, 前 50 年都是通用处理器的发展以绝对优势占据了处理器芯片的市场份额, 相信在接下来的 30 年, 随着数据的爆发和 “端、边、云一体” 这种计算范式的继续渗透, 将形成通用处理器与专用处理器并行的新局面, 在 2020 年 7 月的*Communications of the ACM* 中, 有一篇文章提出了一个新概念: “ASIC clouds”, 全文标题是 “ASIC clouds: specializing the datacenter for planet-scale applications” [8], 这里的 “ASIC” 就是各种专用处理器的呈现形式, 我们相信专用处理器将迎来空前的增长机遇. 笔者认为构建专用处理器系统结构有以下几个关键点.

8.1 针对 “数据平面” 的计算架构

专用计算体系结构和通用计算体系结构的阵地是不同的, 专用计算竞争的焦点是数据平面, 而通用计算竞争的焦点是控制平面. 专用计算像造赛车, 目标是快, 重点是根据赛道的类型来决定赛车的结构; 通用计算像造民用车, 目标更加的多元化, 不仅要兼顾不同路况下的可用性, 还要考虑性价比、代际兼容性. 所以, 以通用 CPU 的标准来看待专用 XPU 可能并不合适, 甚至会制约了专用处理器的

发展.

8.2 融合创新技术

计算架构的范畴不仅仅是狭义的处理器芯片, 还包括相应的存储、传输、集成工艺等, 是一个系统性概念. 专用计算由于其 “专用” 的属性, 对融合新技术更有优势, 例如, 引入高速非易失性存储 (non-volatile memory express, NVMe), 利用 “NVMe oF” 技术构建更高效的分布式存储系统; 将主机内存直接连接在 PCIe 设备端, 建立更大、更快的远程直接内存访问 (remote direct memory access, RDMA); 集成 HBM (high-bandwidth memory) 支持更大的片上数据集, 更高效的数据平面操作; 将神经网络计算融入网内计算, 透明赋能需要推理的场景. 在融合异构核的过程中, 可以采用 “Chiplet” 等提供异质集成的封装技术, 低成本地完成不同功能、不同性能核的集成, 缩短专用处理器的研发周期.

8.3 面向的领域专用描述语言

应用都是可以通过无二意性的语言进行描述的, 专用计算也不例外. 对应用的描述层是专用计算架构的边界: 描述层之上是客户的实际应用程序, 描述层之下都是专用计算系统涉及定制的部分. 整个系统的参考边界由传统 ISA (指令集) 上升到了 DSL (domain-specific language). 例如, P4 编程语言 [15] 是面向 SDN 的领域专用语言, 专门用于定义路由器和交换机如何转发数据包, 属于数据平面的编程语言. 至于网络处理器是用 ARM 还是 MIPS, 或是 X86 并不重要. 现在的深度学习框架例如 TensorFlow, 其实也是提供了一整套定义深度学习模型结构、描述模型训练方法的 DSL; 还有面向数据库的 SQL, 本身就是一种声明式 (declaritive) 的 DSL 编程语言, 有望成为新专用处理器设计的参考边界.

8.4 先垂直深耕, 再水平扩展

对于专用计算架构业界的一个普遍共识是 “碎片化” 问题, 挑战 “one-size-fit-all” 的 ASIC 商业模式. 传统上认为碎片化意味着单个产品线难以上量, 难以摊薄芯片研发的巨额一次性投入, 即所谓的高昂的 NRE (non recurring engineering) 成本. 一个有商业价值的技术必须建立在 “技术闭环” 的基础上: 研发、使用、反馈、再研发改进、再扩大使用范围 ..... 技术只有投入使用才能体现价值, 有使用价值才有可能商业化, 才能完成技术闭环到商业闭环的进化. 技术闭环的形成需要集中火力, 全链条主动出击才能铺就. 应用 “碎片化” 并不是 “专用” 障碍, 反而是专用处理器技术路线应该充分利用的优势.

9 结论

采用专用处理器芯片是计算系统提升算力、提高效率的有效手段, 业界对于 “XPU” 概念的广泛关注反映了人们对新型计算芯片期望. 本文以经典的数字信号处理器 DSP、图形处理器 GPU、AI 芯片、网络处理器 (NPU) 作为案例, 以这 4 类专用处理器的结构特征为分析重点, 总结得出 DSP 最重要的结构特征是支持灵活的数据格式, 能效比优化极致; GPU 是充分利用数据并行的典型, 并且正在朝着通用化的方向发展; AI 芯片围绕大规模张量处理, 支持可变精度来优化性能; NPU 聚焦网络数据包处理, 构造高通量的流水线. 笔者发现一些在通用 CPU 中没有得到成功发展的技术, 如 VLIW, 在专用处理器中发挥了重要作用; 而通用处理器体系下最重要维度, 如指令系统, 在专用处理器中反而被弱化了. 这些差异进一步引发了笔者从经典体系结构的角度对专用处理器的几点思考. 最后, 本文

讨论构建专用处理器系统结构的 4 个关键点: 即针对数据平面的架构, 融合新存储、传输、封装等新技术, 面向领域专用语言和充分利用好 “专用” 这个特征来简化系统设计.

参考文献

1 Hennessy J L, Patterson D A. A new golden age for computer architecture. Commun ACM, 2019, 62: 48–60
2 Walravens C, Dehaene W. Low-power digital signal processor architecture for wireless sensor nodes. IEEE Trans VLSI Syst, 2014, 22: 313–321
3 George L, Blume M. Taming the IXP network processor. SIGPLAN Not, 2003, 38: 26–37
4 Lindholm E, Nickolls J, Oberman S, et al. NVIDIA Tesla: a unified graphics and computing architecture. IEEE Micro, 2008, 28: 39–55
5 Chen T S, Du Z D, Sun N H, et al. DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning. SIGARCH Comput Archit News, 2014, 42: 269–284
6 Jouppi N P, Young C, Patil N, et al. In-datacenter performance analysis of a tensor processing unit. In: Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA), 2017
7 Yan G H, Lu W Y, Li X W, et al. DPU: data-centric domain-specific processor. Commun CCF, 2021, 17: 51–56 [鄢贵海, 卢文岩, 李晓维, 等. DPU: 以数据为中心的专用处理器. 中国计算机学会通讯, 2021, 17: 51–56]
8 Taylor M B, Vega L, Khazraee M, et al. ASIC clouds: specializing the datacenter for planet-scale applications. Commun ACM, 2020, 63: 103–109
9 Stone J E, Gohara D, Shi G. OpenCL: a parallel programming standard for heterogeneous computing systems. Comput Sci Eng, 2010, 12: 66–73
10 Chen Y J, Luo T, Liu S L, et al. DaDianNao: a machine-learning supercomputer. In: Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, 2014. 609–622
11 Lu W Y, Yan G H, Li J J, et al. FlexFlow: a flexible dataflow accelerator architecture for convolutional neural networks. In: Proceedings of IEEE International Symposium on High Performance Computer Architecture (HPCA), 2017. 553–564
12 Lu W Y, Yan G H, Li J J, et al. Promoting the Harmony between sparsity and regularity: a relaxed synchronous architecture for convolutional neural networks. IEEE Trans Comput, 2019, 68: 867–881
13 Zimmermann H. OSI reference model — the ISO model of architecture for open systems interconnection. IEEE Trans Commun, 1980, 28: 425–432
14 Tsai Y M, Cojean T, Anzt H. Evaluating the performance of NVIDIA’s A100 ampere GPU for sparse linear algebra computations. 2020. ArXiv:2008.08478
15 Bosshart P, Daly D, Gibb G, et al. P4: programming protocol-independent packet processors. SIGCOMM Comput Commun Rev, 2014, 44: 87–95

Comparative study of the domain-specific processors

Guihai YAN*, Wenyan LU, Xiaowei LI & Ninghui SUN

State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190*, China*
* Corresponding author. E-mail: yan@ict.ac.cn

Abstract Microprocessor is the core component in modern information systems. The explosive growth of data volume, fueled by the rapidly evolved technology including big data, artificial intelligence and 5G, demands an unprecedented high computing capability. It has been believed that domain-specific computing will be one of the major directions of computer architecture in the post-Moore's Law era. The evolution of domain-specific processors is actually always accompanied by general-purpose processors. Digital signal processor (DSP), one of the typical domain-specific architectures, appeared even earlier than common CPU technology. General-purpose computing, which has been witnessed to achieve huge commercial success, also shares lots of key technologies to boost domain-specific architectures, in terms of performance and programmability. By surveying representative commercial products of DSP, GPU, AI chips and NPU, this paper aims to explore the key architectural features of these domain-specific architectures, and is supposed to shed some light on the future research and development of domain-specific computing technology.

Keywords domain-specific processor, digital signal processing, graphic processing, deep learning, network processing