中国科学 : 信息科学 2025 年 第 55 卷 第 3 期: 452–463
SCIENTIA SINICA Informationis
《中国科学》杂志社
SCIENCE CHINA PRESS
先进计算与新兴软件专题 · 论文

数据流芯片的发展现状、趋势与挑战

冷静文[1], 过敏意[1*], 曾德泽[2], 蒋文斌[3], 叶笑春[4], 陈华曦[5], 李文明[4]

1. 上海交通大学电子信息与电气工程学院, 上海 200240
2. 中国地质大学 (武汉) 计算机学院, 武汉 430074
3. 华中科技大学计算机科学与技术学院, 武汉 430074
4. 中国科学院计算技术研究所处理器芯片全国重点实验室, 北京 100190
5. 之江实验室智能计算平台中心, 杭州 311121
* 通信作者. E-mail: guo-my@cs.sjtu.edu.cn

收稿日期: 2024–11–13; 修回日期: 2025–01–09; 接受日期: 2025–02–18; 网络出版日期: 2025–02–27

国家重点研发计划 (批准号: 2022YFB4501400) 资助项目

摘要 本文聚焦于新型数据流架构在多领域融合计算中的应用潜力与发展趋势. 随着人工智能、图计算和大数据等新兴技术的发展, 传统冯 · 诺依曼 (von Neumann) 架构和领域专用架构的性能瓶颈愈加显著, 难以满足未来计算系统对高性能和高灵活性的需求. 本文首先梳理了现有数据流芯片的设计方法, 基于专用性与通用性、执行粒度等维度探讨了数据流架构的不同实现方式及其应用现状. 在此基础上, 提出了一种基于并发代码块的数据流抽象机模型, 并设计了完整的指令集和微体系结构, 进一步实现了跨领域的统一中间表示和多种算子融合策略, 优化了数据流在图神经网络、大模型计算和实时信号处理等任务中的执行效率. 实验结果表明, 基于该抽象机模型的处理器在性能与功耗方面优于现有的通用处理器架构. 最终, 本文展望了数据流架构在未来计算系统中的广泛应用前景及其对高效能计算的深远影响.

关键词 多领域融合计算, 数据流架构, 抽象机模型

1 引言

计算机技术在自然语言与视觉处理、信息检索、个性化推荐等方面不断发展和突破, 在国计民生的各个经济和社会活动领域应用愈加广泛; 这也对计算机系统的性能、功耗和易用性提出更高的需求. 通用 CPU (central processing unit) 处理器受制于冯 · 诺依曼 (von Neumann) 控制流架构的瓶颈制约, 性能功耗已逐渐饱和; 在经典的乱序执行 CPU 处理器中, 用于计算部分的功耗仅占 6% [1], 用于指令调度的开销接近整体功耗的 50%. 为此, 英伟达的通用 GPU (graphics processing unit) 处理器采用超多线程的并发调度方式, 极大地减轻了冯氏架构中指令开销, 计算部分的功耗可提升至 20% 以上 [2]. 更进一步的, 随着人工智能 (artificial intelligence, AI) 技术的快速发展, 处理器由通用架构转变为领域

引用格式: 冷静文, 过敏意, 曾德泽, 等. 数据流芯片的发展现状、趋势与挑战. 中国科学: 信息科学, 2025, 55: 452–463, doi: 10.1360/SSI-2024-0343
Leng J W, Guo M Y, Zeng D Z, et al. Dataflow microprocessor: development, trends, and challenges. Sci Sin Inform, 2025, 55: 452–463, doi: 10.1360/SSI-2024-0343

专用架构, 如谷歌深度学习专用处理器 TPU (tensor processing unit) [3] 以及英伟达在 GPU 基础上集成的张量单元 TensorCore [4]. 然而, AI 技术呈现多领域交叉融合的发展趋势, 例如 AI 与图计算和大数据结合, 分别产生了图神经网络和向量型数据库等多领域融合负载. 该发展趋势给灵活性受限的领域专用架构带来了全新的挑战: 如何突破通用 CPU/GPU 处理器的瓶颈, 构建性能高、效能高、灵活性高的新型计算机处理器是目前系统结构领域的研究重点, 也是工业界关注的焦点.

数据流架构是构建高性能与高效能处理器的重要方法. 数据流具有天然的数据驱动并行执行模型, 能够有效地驱动大量计算和存储资源. 例如, 谷歌深度学习专用处理器 TPU [3] 以及英伟达 GPU 中的 TensorCore [4] 均可被认为是一种特殊固化后的数据流架构. 此外, 数据流模型也具有可完整表示应用计算语义、提供简洁编程性等优势; 深度学习领域中如 TensorFlow [5] 与 PyTorch [6] 等编程框架和 TVM [7] 等编译框架都采取了基于数据流的计算模型, 获得了广泛使用. 然而, 数据流芯片目前只在深度学习领域取得了初步的成功, 其硬件架构和配套软件系统无法用于其他领域, 更遑论新兴多领域融合负载. 因此, 亟须研究新型数据流芯片以及计算系统. 在此背景下, 国家重点研发计划在 “先进计算与新兴软件” 重点专项中部署了 “新型数据流异构处理器架构及计算系统” 项目, 重点攻关面向多领域融合计算的新型芯片架构设计以及研制. 该项目由上海交通大学过敏意教授作为项目总负责人, 联合阿里云、华中科技大学、中国科学院计算技术研究所、之江实验室等课题单位, 目前该项目已经顺利通过中期检查.

本研究团队针对多领域融合计算存在数据 – 控制 – 访存密集的时空交织复合特征, 给单一数据流或者控制流架构的现有计算机系统带来了数据频繁交互导致性能低、编程效率与可用性差、资源分配固化利用率低下等系列挑战, 提出了基于并发代码块的数据流抽象机, 并在此抽象机的基础上, 设计了完整的指令集以及微体系结构. 通过 FPGA (field-programmable gate array) 的仿真评估, 该处理器可在落后两代芯片制程的情况下, 取得比英伟达 Orin 处理器 [8] 更好的性能以及功耗. 在计算生态上, 该芯片可以兼容 CUDA (compute unified devices architectured) 的单指令多线程 (single instruction multiple thread, SIMT) 模型, 并在此基础上进一步支持数据流计算模型, 提高计算效率. 本文首先介绍数据流芯片的研究现状以及发展趋势, 进一步描述数据流芯片的计算生态挑战, 最后介绍本研究团队中所提出的并发代码块数据流抽象机以及相应的芯片架构设计.

2　数据流芯片的研究现状与发展趋势

传统冯 · 诺依曼计算机以控制流计算模型为基础, 其执行按照程序顺序执行; 而数据流芯片将程序表示为一个有向图, 其按照有向图中的节点顺序来执行. 本文通过两个维度对数据流芯片的现状和发展趋势进行介绍. 其中, 第一个维度是芯片的专用性与通用性, 第二个维度是数据流的执行粒度, 也就是程序对应的有向图中的节点粒度.

根据第一个维度, 数据流芯片的架构可以分为专用数据流架构与通用数据流架构. 在专用数据流架构中, 芯片将特定的程序所对应的数据流图直接展开并映射在硬件上, 达到很高的执行效率; 然而其仅仅支持特定的程序, 因此称为专用数据流芯片架构. 目前专用数据流芯片架构代表包括面向矩阵乘法的谷歌深度学习专用处理器 TPU [3] 以及英伟达 GPU 中的 TensorCore [4].

根据第二个维度, 数据流芯片的执行粒度由细到粗, 如图 1 所示, 可以分为比特级、操作数级、指令级、线程级. 不同的数据流粒度决定了程序的数据流图中每个节点的粒度大小, 这对架构中的并行性、复杂度、编程生态方面有着重要影响. 通常来说, 较细粒度的架构 (如比特级、操作数级) 可以实现更高的并行性和更精确的资源控制, 但相应的硬件开销较大且调度复杂度也更高. 而较粗粒度的架构 (如指令级、线程级) 在并行性上可能会有所折中, 但通常能更好地适应多种应用场景, 具有较好的扩展性.

比特级数据流架构. 该架构的主要代表为 FPGA, 其执行过程中按照比特操作的数据流图进行计

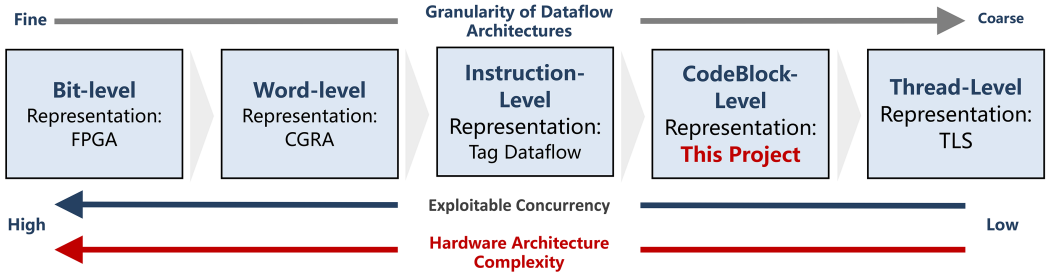

图 1　(网络版彩图) 数据流架构的粒度以及硬件复杂度对比.

Figure 1　(Color online) Comparison of data flow architecture granularity and hardware complexity.

算. 这种方式实现了极高的并行性, 但硬件资源需求非常高, 且数据通路设计较为复杂. 此外, FPGA 的编程通常需要使用硬件描述语言 (如 Verilog 或 VHDL), 其设计复杂度和门槛远高于一般的软件编程, 开发周期较长. 此外, 调试过程也较为复杂, 需要硬件级的工具和流程; 其也需要在每次上电后进行配置, 这会带来启动延迟, 并且在编程和配置过程中可能会占用较多的时间, 限制了实时性要求高的应用.

操作数级数据流架构. 该架构的主要代表为 CGRA (coarse-grained reconfigurable architecture), 其将单个操作数作为处理的基本单元, 即每次执行一个操作数的计算节点. 相比比特级架构, 操作数级架构在硬件成本上有所降低, 同时仍能保持较高的并行性. 斯坦福大学 (Stanford University) 的初创公司 SambaNova 的可重构数据流处理器 (reconfigurable dataflow unit, RDU) [8] 是一种操作数级的数据流架构, 可以高效执行机器学习和高性能计算任务. 在这种架构中, 程序被表示为数据流图, 其中每个节点对应于操作数级的计算单元. RDU 通过将这些节点直接映射到硬件, 实现高度并行的计算, 从而提高性能和效率. RDU 的核心组件包括模式计算单元 (pattern compute unit, PCU) 和模式存储单元 (pattern memory unit, PMU). PCU 负责执行单个操作数级的计算操作, 采用多阶段、可重构的 SIMD 流水线设计, 支持高计算密度. PMU 则提供片上存储容量和带宽, 支持灵活的数据访问模式, 减少数据移动和延迟. 这些组件通过高速的片上交换网络连接, 形成一个可重构的功能单元网络, 能够根据不同的应用需求进行动态配置. 与传统的指令集架构不同, RDU 没有固定的指令集, 而是为每个模型专门编程, 生成高度优化的、特定于应用的加速器. 这种操作数级的数据流架构通过减少冗余的数据移动和指令处理开销, 实现了更高的硬件利用率和性能.

指令级数据流架构. 指令级数据流架构以德克萨斯大学奥斯汀分校 (University of Texas at Austin) 的 TRIPS [9] 和华盛顿大学 (University of Washington) 的 WaveScalar [10] 为代表. 在指令级数据流架构中, 每条指令被视为一个独立的计算节点, 构成一个指令流图, 节点之间的数据依赖关系决定了执行顺序. 这种架构的优势在于其良好的通用性和适用于多种负载的能力, 特别是在任务并行化和动态调度方面具有优势. TRIPS 架构 [9] 是一种指令级的数据流架构, 能够支持多达 128 条指令的并行执行. TRIPS 通过一系列控制单元和分布式存储单元, 将指令以图的形式分发到多个处理单元中进行计算. WaveScalar 架构则是一个基于指令级数据流的动态计算模型, 其通过引入波流 (wavefront) 的概念, 将指令按波的形式推进执行, 这种方式在执行过程中自动检测并维护指令之间的数据依赖. 尽管这些指令级数据流架构在某些负载上取得了很高的执行效率, 其在与基于控制流的冯氏 CPU 架构的竞争中处于下风, 其原因在于对于通用计算的适配性较低以及在复杂计算模式下无法保持较高的计算效率. 此外, 数据流架构中充分挖掘细粒度并行度的思想在如今高性能 CPU 中得以吸收保留, 演变成了如乱序执行, 超标量发射等技术.

线程级数据流. 线程级数据流主要为软件概念, 其代表包括 CPU 架构上的线程级流水 (thread-level pipelining) [11] 以及 GPU 架构上的线程束专用化 (warp specialization) [12]. 线程级数据流架构主

要以线程为基本计算单元, 通常不依赖硬件中的细粒度控制, 而是通过软件层面的调度来实现资源分配和并行度控制. 在传统的 CPU 架构中, 线程级流水 (thread-level pipelining) [11] 是一种典型的线程级数据流概念. 线程级流水通过在不同线程之间分配独立的流水线阶段, 使得多个线程能够并发执行而不会相互阻塞. 每个线程在各自的流水线阶段中独立工作, 从而实现了并行计算和资源的高效利用. 线程级流水在处理多任务、多线程应用时表现尤为出色, 尤其适合于数据库处理、网络通信等并行任务需求较高的场景. 在 GPU 架构中, 线程束专用化 (warp specialization) [12] 将不同的线程束按照数据流图的依赖关系进行编排, 相较于 GPU 中默认的线程无序并发竞争硬件资源的 SIMT 执行模型, 线程束专用化可以取得更好的执行效率, 例如高性能大模型算子库 FlashAttention-3 [13] 中就采用了线程束专用化模型以发挥 H100 GPU 的算力.

数据流架构也可以基于上述不同的粒度进行组合, 例如 Groq 公司的 LPU 处理器 [14] 的架构组合了操作数级数据流与指令级数据流. 具体而言, 操作数 (数据) 可以在 LPU (language processing unit) 中横向流动, 而指令可以在其中纵向流动. 然而, 其操作数与指令的流动节拍都需要程序预先设定好, 且在运行期间不可修改. 这也导致了 LPU 处理器架构无法采用任何延迟可变的被动性部件, 例如常见的缓存以及片下 DRAM (dynamic random-access memory), 仅能使用延迟固定的部件, 如片上的 SRAM 等. 在程序的存储需求 (如大模型 LLM (large language model)) 超过其片上 SRAM (static random-access memory) 的容量时, 需要扩展多个 LPU; 其导致的成本问题以及跨芯片互联问题使得其应用范围较窄.

3　数据流芯片的计算生态挑战

通过分析上述可以发现, 过去及现有的数据流架构设计主要以性能优化为核心, 虽然在特定任务上能够显著超越传统的 CPU 和 GPU 架构, 但在计算生态的适配性上存在明显短板. 这些架构往往需要专门的编程模型和工具链支持, 且硬件设计的定制化和专用性限制了其通用性和灵活性. 与此同时, CPU 和 GPU 架构在逐步吸收数据流计算的核心思想, 并在架构和编程模型上不断演进以适应更广泛的计算需求. 例如, CPU 架构在乱序执行、超标量技术、线程级流水线上集成了数据流的思想, 以优化指令调度和资源利用率; 而 GPU 架构则通过引入 TensorCore 硬件部件和线程束专用化流水线模型 (如 Warp Specialization), 显著提升了深度学习和高并行计算任务的性能和效率. 通过对数据流思想的融合, CPU 和 GPU 架构不仅提升了硬件性能, 还能依赖现有的编程生态系统, 在通用性和灵活性方面实现更高的扩展性和适用性. 因此, 未来的数据流架构设计若要真正取得广泛应用, 需要在性能和生态适配性之间找到更好的平衡. 本文总结了如下两个具体的计算生态构建方面的挑战.

关键挑战一: 硬件的抽象机模型. 传统的 CPU 架构能够以线程模型作为抽象机模型, 这意味着程序员可以以线程为基本单位, 通过调度和并行化机制来高效利用硬件资源. 而在 GPU 架构中, 通过单指令多线程 (SIMT) 模型提供了一个更高的并发维度, 使得程序员在串行思维的基础上, 使用简单的语法扩展即可发射大量的并发线程, 从而充分发挥出硬件的计算潜力. GPU 的抽象机模型的设计不仅简化了编程流程, 还增强了架构的适用性和通用性, 使得其在新兴的计算机应用中获得远超 CPU 的使用. 因此, 数据流架构若想实现广泛应用, 如何设计一个既具备高效并行性又易于编程的抽象机模型便成为了关键挑战之一. 只有当数据流架构拥有一个能够简化编程、增强可移植性并能高效利用硬件资源的抽象模型时, 才能真正推动其在广泛计算场景中的普及和应用.

关键挑战二: 数据流架构的粒度选择. 正如前文所述, 数据流芯片的架构粒度从比特级、操作数级到指令级、线程级, 粒度越细, 编程难度越高. 比特级和操作数级的数据流架构, 如 FPGA 和 CGRA (例如 SambaNova 的 RDU 处理器), 在编程过程中需要与底层硬件深度绑定, 导致编程门槛较高, 难以构建适配广泛应用的计算生态. 而指令级数据流架构 (如德克萨斯大学奥斯汀分校的 TRIPS 和华盛顿大学的 WaveScalar) 虽然可以直接支持 C 语言等高级编程语言, 但对编译技术的要求极高, 同时

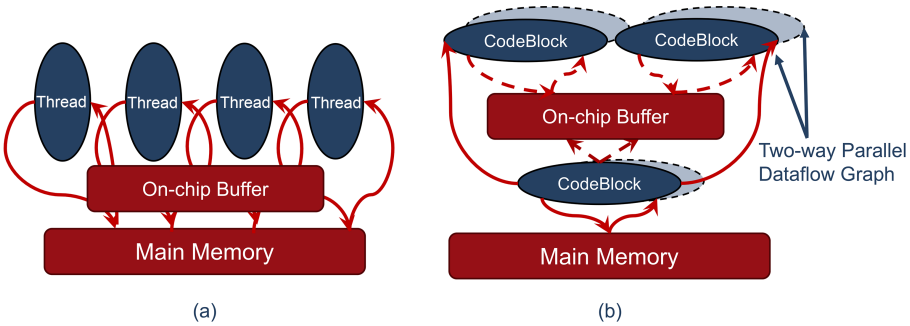

图 2　(网络版彩图) 编程模型对比. (a) GPU 的单指令多线程 SIMT 模型; (b) 本项目的并发代码块数据流模型.

Figure 2　(Color online) Comparison of programming models. (a) Single instruction, multiple threads (SIMT) model of a GPU; (b) concurrent codeblock dataflow model of this project.

CPU 架构在指令级的优化方面已经相当成熟, 因此指令级数据流架构在通用性上的竞争优势有限. 线程级数据流则已经在传统的 CPU 和 GPU 架构中得到了良好的支持, 且不需要硬件上的额外改动, 具备广泛的适用性. 因此, 对于新型的数据流架构来说, 如果要在性能与生态适配性上取得平衡, 可能需要探索一种介于指令级和线程级之间的粒度, 既能够提供较高的并行性, 又能兼顾编程友好性和硬件适配性, 从而推动数据流架构在多种计算场景中的广泛应用.

4　基于并发代码块数据流抽象机的芯片设计

针对上述挑战, 本研究团队提出了基于并发代码块数据流的抽象机模型, 并根据该抽象机模型设计了相应的指令集以及微体系结构, 规划了相应的软件生态构建方案, 有望突破现有 GPU 的计算效率并兼顾其生态, 支撑未来的多领域融合应用.

4.1　并发代码块数据流抽象机模型

如前文所论述的, 为推动新型数据流架构在多种计算场景中的广泛应用, 需要探索一种介于指令级和线程级之间的数据流粒度, 既能够提供较高的并行性, 又能兼顾编程友好性和硬件. 为此, 本研究团队借鉴了数据流先驱高光荣先生提出来的 Codelet 数据流抽象机模型 [15]. 在这个模型中, 计算被分解为称为 “Codelet” 的小型、独立的单元, 这些单元根据数据的可用性进行调度和执行. 该抽象机模型一开始是为分布式系统的超算所设计的 [16], 对于硬件架构的设计并无指导性的方案.

为此, 本研究团队提出了一种基于代码块的数据流抽象机模型, 如图 2 所示. 在该模型中, 数据流图的基本粒度被定义为由多条指令组成的代码块. 每个代码块内部的指令不含分支控制指令, 所有指令按照顺序串行执行, 简化了执行控制逻辑, 降低了编程复杂度. 在代码块与代码块之间, 则通过指定数据依赖关系建立细粒度的生产者 – 消费者关系, 从而实现高度灵活的并行调度. 这种设计不仅使得代码块的并发执行更加高效, 同时为编程人员提供了简洁的抽象模型, 可以在一定程度上减少硬件调度和资源管理的复杂性, 为构建高效的数据流计算生态提供了重要基础.

在上述代码块数据流抽象机模型的基础上, 本研究团队创新性地融合了 GPU 抽象机模型中的并发流概念, 构建了基于并发代码块的数据流抽象机模型, 从而实现与 GPU 兼容的软件生态. 在该扩展后的模型中, 用户可以借鉴单指令多线程 (SIMT) 模型的编程方式, 通过类似的简单语法对代码块进行并发扩展. 这样一来, 开发者可以像在 GPU 上进行多线程编程一样, 通过简洁的语法轻松地控制并发操作, 从而降低编程门槛, 同时提高硬件的资源利用率和执行效率. 该模型为程序员提供了熟悉的编程体验, 并能与现有的 GPU 编程生态无缝衔接, 为数据流架构的广泛应用铺平了道路.

如表 1 所示, 在兼容 GPU 的单指令多线程 (SIMT) 模型的同时, 本研究团队的并发代码块数据流抽象机模型在性能和效率上具有显著优势. 首先, 在调度开销方面, 本研究团队的抽象机模型中代码

表 1　各种架构的抽象机模型和关键技术参数对比. 粗体表示最优结果.

Table 1 Comparison of abstract machine models and key technical parameters across different architectures. The best results are in bold.

		GPU	LPU	DFU
Programming abstraction comparison	Abstract machine	Control flow (single instruction multiple thread, SMIT)	–	Concurrent dataflow
	Ecosystem openness	Virtual ISA	–	Virtual ISA
	Programming model	CUDA	–	DFUC
Technical parameter comparison	Concurrency	Multi-thread	–	Multi-graph
	Scheduling method	Streaming	Preset	Dataflow graph
	Bandwidth requirement	High	**Low**	**Low**
	Memory access latency	High	**Low**	**Low**
	DRAM scalability	**Yes**	No (limited by preset scheduling, not support DRAM)	**Yes**

块的存储开销远低于 SIMT 模型中的线程存储开销, 几乎可以忽略不计. 与每个线程需维护私有寄存器不同, 所有代码块共享寄存器资源, 而在 GPU 上, 由于大量线程的并行执行, 寄存器的开销往往占整体功耗的 20% 左右. 根据之前相关工作的估算 [17], 如果能实现 GPU 架构中寄存器的有效共享, 在运行深度学习模型时可提升超过 30% 的性能以及节省近 30% 的能耗. 其次, 由于代码块的创建开销极低, 其并发数量可根据计算需求动态调整, 从而更灵活地适应复杂多变的并发要求. 而在 SIMT 模型中, 线程创建开销较高, 因此线程数量在一个 kernel 内核函数内难以动态调整, 通常需要在不同内核函数中变化, 这不仅增加了编程复杂性, 也带来了额外的 CPU 调度开销. 最后, SIMT 模型中的线程在执行时会无序竞争硬件资源, 这往往增加了存储和带宽需求; 而代码块则按照数据流图有序执行, 对硬件资源的需求相对更低, 从而实现了更高效的资源利用率和更优的功耗性能. 根据 PyTorch 官方的报告, 在 H100 芯片上, 使用线程束数据流的方式运行矩阵运算, 相较于使用传统的 SIMT 模型执行方式, 芯片利用率可提升一倍以上 [18].

4.2　并发代码块数据流的指令集

针对上述并发代码块的数据流抽象机模型, 本研究团队设计了一套动态融合数据流与控制流的超精简指令集, 称作 VeryRISC. VeryRISC 指令集充当了 DFU (dataflow unit) 硬件芯片与软件之间的接口, 使得汇编编程者与编译器可以无缝地并发使用两种流式计算部件. VeryRISC 指令集采用了高度模块化的设计思路, 包含计算指令、访存指令和数据流指令 3 个部分. 计算、访存指令集针对代码块内部的数据计算和内存读写操作进行了精细化设计, 涵盖了数据加载、存储、算术运算等常见操作. 这些指令为代码块内的计算任务提供了高效的实现路径, 特别是在处理并发任务时, 可以显著减少上下文切换和资源竞争, 提高运算性能. 数据流指令集则以代码块的具体应用场景为核心, 专门设计了通过片上互联进行 PE 间寄存器点对点传输的 Copy 指令, 使代码块能够进行高效的数据复用以适应更多样化的计算任务.

在硬件实现方面, VeryRISC 指令集充分考虑了代码块内部计算的特性, 通过精简解码逻辑和优化数据路径设计, 降低了芯片面积和功耗. VeryRISC 采用了固定长度的指令格式, 保证了指令解码的一致性, 从而简化了指令流水线的设计. 为了进一步提升代码块内运算的高效性, VeryRISC 对内存访问和算术运算的指令执行进行了专门优化, 使得在片上资源有限的情况下, 也能实现低延迟的指令处理.

VeryRISC 指令集的具体指令格式如表 2 所示, 它仅包含 11 条固定长度的基本指令. 所有指令都

表 2　VeryRISC 指令集格式.

Table 2　VeryRISC instruction set format.

Stage	Operation	Main function
LOAD	LD	OPM[F0] = DRAM[LD_Base + {F1, F2}]
	ADD	OPM[F2] = OPM[F0] + OPM[F1]
	SUB	OPM[F2] = OPM[F0] − OPM[F1]
	MUL	OPM[F2] = OPM[F0]×OPM[F1]
	MAX	OPM[F2] = MAX(OPM[F0], OPM[F1])
CAL	MIN	OPM[F2] = MIN(OPM[F0], OPM[F1])
	MADD	OPM[F2] = (OPM[F0]×OPM[F1]) + OPM[F2]
	PREREAD0	OP0-PreRead Data Reg = OPM[F0]
	PREREAD1	OP0-PreRead Data Reg = OPM[F1]
FLOW	COPY	PE[F2].OPM[F1] = OPM[F0]
STORE	ST	DRAM[ST_Base + {F1, F2}] = OPM[F0]

有相同的格式: [OP F0, F1, F2, CTRL]. 其中 OP 是 4 位指令类型字段, F0 ∼ F2 是 3 个 16 位操作数字段, CTRL 是一个 12 位控制字段, 带有一个 8 位的稀疏 PC 增量子字段, 以支持稀疏神经网络. 以及一个 4 位表示 DRAM 内查找类型的子字段, 以支持复杂的激活/分类器功能. 具体的指令描述如下.

(1) LD 指令 (加载指令). LD 指令 (专注于将数据从主存 (DRAM) 加载到片上内存 (scratchpad memory, SPM)) 是数据流计算的起点. LD 指令通过基址寄存器 (LD_Base) 和两个偏移寄存器 (F1, F2) 灵活定位数据地址, 将主存中的数据准确加载到指定的片上内存寄存器 (SPM[F0]). 这种基址加偏移寻址方式不仅支持顺序访问, 还能够满足复杂的非连续数据加载需求, 大幅提高了访存效率, 为后续的高效计算奠定了基础.

(2) CAL 指令 (计算指令). CAL 指令是该指令集的核心部分, 提供了多种算术和逻辑操作指令, 覆盖了高性能计算所需的常用操作, 包括算术逻辑运算: 如加法 (ADD)、减法 (SUB)、乘法 (MUL) 等, 用于执行基本的数值计算; 逻辑与比较: 如最大值 (MAX)、最小值 (MIN) 等操作, 可高效地进行条件选择; 复杂运算: 乘累加 (MADD) 指令在单条指令中完成乘法与加法的组合操作, 适合矩阵运算和信号处理等场景. 此外, 该阶段还引入了两个预读取指令 (PREREAD0 和 PREREAD1), 用于提前将数据加载到特定寄存器中, 避免访存延迟对计算流水线的影响. 这种设计充分考虑了代码块内部高频计算和访存的并发需求, 确保了计算的持续性和高效性.

(3) FLOW/Copy 指令 (流动指令). FLOW/Copy 指令实现片上处理单元 (processing element, PE) 之间的数据交换, 是数据流计算中不可或缺的一部分. Copy 指令支持从源处理单元 (PE[F0]) 将数据复制到目标处理单元 (PE[F2].SPM[F1]). 这种高效的数据流动机制能够减少代码块内部的通信开销, 充分发挥多处理单元并行计算的能力.

(4) ST 指令 (存储指令). ST 指令专注于将片上内存中的数据存储回主存. 通过 ST 指令, 开发者可以将计算结果从片上内存 (SPM[F0]) 写入主存 (DRAM), 并支持基址 (ST_Base) 加偏移 (F1, F2) 的灵活存储地址计算. 这种设计确保了在多代码块的协同处理中, 数据能够被高效地存储和共享, 同时减少主存操作的延迟.

4.3　并发代码块数据流的芯片设计

本研究团队的并发代码块数据流芯片包含 16 个处理单元 (PE). 每个处理单元均支持 64 位 VeryRISC 超精简指令集. 该指令集允许通过 Copy 指令将一个 PE 的向量寄存器中的数据直接通过片上网络拷贝到另一个 PE 的向量寄存器中, 从而显著提高了数据流动的灵活性和效率.

表 3　数据流芯片与 Orin GPU 硬件参数对比.

Table 3　Comparison of hardware specifications between dataflow chips and Orin GPUs.

Metric	Dataflow Chip DFU2.0	Orin CPU
Process	12 nm	8 nm
CPU cores	16-core RISC-V @ 1.5∼2 GHz	8-core ARM @ 2.2 GHz
Coprocessor	Dataflow core array	2 NVDLA accelerators
Memory bandwidth	200 GB/s	200 GB/s
FP8 compute power	80 TOPS INT8, 80 TFLOPS	100 TOPS INT8
FP16 compute power	40 TFLOPS FP16	42.5 TFLOPS FP16
FP32 compute power	5 TFLOPS FP32	5.32 TFLOPS FP32
FP64 compute power	2.5 TFLOPS FP64	–
Interconnect	PCIe4.0 32 GB/s	PCIe4.0 32 GB/s
Power consumption	75 W	60 W

在本架构中, 每个处理单元独立维护自己的代码块. 代码块实际上是一个串行的指令流, 负责执行特定的计算任务. 由于存在 Copy 指令, 各个代码块之间形成了复杂的依赖关系. 具体而言, 下游的代码块必须在其上游代码块完成执行后才能启动. 这种依赖关系的管理并非由软件层面处理, 而是由硬件自动维护, 确保在多处理环境中数据的正确性和执行的顺序性.

相比之下, Orin GPU 则采用了传统的并行处理架构, 主要依靠多线程并行和共享存储器来管理数据传输. 数据流芯片与 Orin GPU 的硬件参数对比如表 3 所示. Orin 的架构在处理高吞吐量、大规模并行计算任务方面表现出色. 然而, 在需要频繁和低延迟的数据交互的任务场景中, Orin 的多线程同步机制和共享存储器访问模式可能带来额外的延迟和资源消耗, 尤其是在数据依赖较强的任务中, 数据传输效率可能受到影响.

DFU2.0 数据流芯片在架构上更强调处理单元之间的灵活协作和低延迟的数据传递能力. 由于每个 PE 独立维护自己的代码块, 并通过硬件级别的自动依赖管理来确保上下游代码块的执行顺序和数据正确性. 因此, DFU2.0 在高度依赖数据的计算任务中可以有效降低同步成本, 并实现高效的并发处理. 此外, DFU2.0 的数据流模式还具有较强的可扩展性, 能够通过扩展 PE 数量来适应不同规模的并发任务需求.

4.4　并发代码块数据流软件生态构建

基于上述抽象机模型及其对应的芯片实现, 本研究团队将构建如图 3 所示的既兼容 CUDA 生态、又具备自主优势的数据流体系结构生态. 选择兼容 CUDA 的原因是其作为当前最为广泛使用的并行计算生态, 同时也是目前摩尔线程、燧原科技等国产芯片公司的对接接口. CUDA 生态采用基于并发线程的抽象机模型, 可认为是并发代码块的一个特例, 即将整个代码块映射至一个线程. 因此, 本研究团队计划构建的软件计算生态从理论上可保证兼容 CUDA 生态.

在兼容 CUDA 生态的基础上, 本研究团队也将开发数据流体系结构特有的基础软件, 以充分发挥数据流架构的性能优势. 具体而言, 数据流芯片的最大优势在于图算融合, 而 GPU 等控制流芯片只能做到图算分离, 即计算图是按照数据流的方式调度, 但是每个算子内部按照控制流的方式来调度. 这也使得 GPU 上需要依赖例如 FlashAttention 等手工算子融合的方式来提升性能. 相比而言, 在数据流架构上, 我们需将算子级的计算图进一步拆解成细粒度数据块 (tile) 级别的数据流图, 依靠硬件自有的数据流调度特性完成执行. 在整个软件中, 只有最后一个粒度的数据流图执行与底层硬件是强绑定的, 其他部分可以与 Ascend 等处理器共同使用, 而与硬件绑定的部分可在特定架构上通过软件模拟的方式来完成.

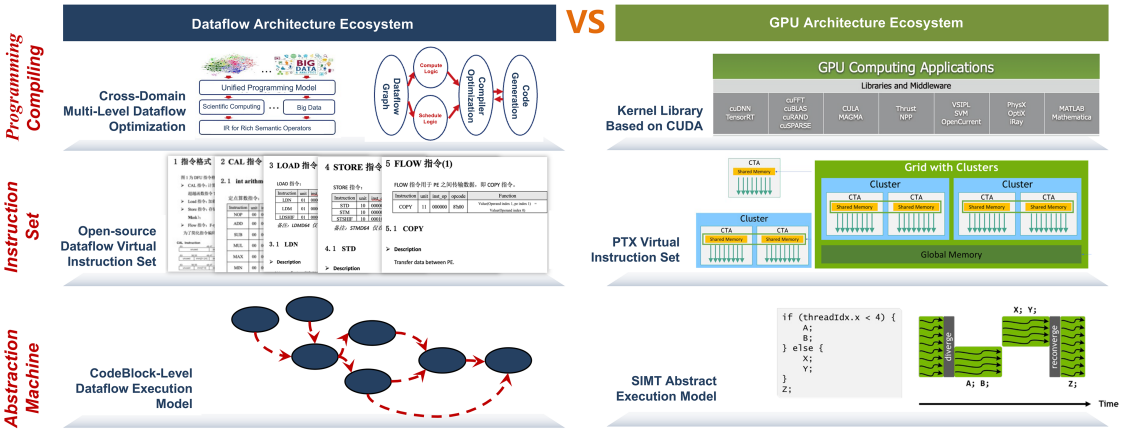

图 3　(网络版彩图) 建设既兼容 CUDA 又有自主优势的数据流体系结构生态.

Figure 3　(Color online) Building a dataflow architecture ecosystem that is both CUDA-compatible and independent.

基于数据流图作为各领域计算模式的统一抽象表达这一特性, 本研究团队为人工智能、科学计算、图计算和大数据等 4 个领域设计了一种基于 MLIR 的跨领域统一中间表示. 在此统一中间表示的基础上, 本研究团队深入挖掘了数据复用关系, 对代码块级的数据流编译进行了优化, 充分利用数据流在数据传输上的优势, 从而提升整体计算性能. 针对异构数据流设备的运行特点, 本研究团队研究了多种算子融合方式, 以降低运行开销并有效提升计算效率. 具体而言, 本研究团队实现了以下 3 种算子融合策略: 首先是纵向算子融合, 即融合在一条计算路径中存在数据依赖的两个算子, 以减少中间数据的传输成本; 其次是横向算子融合, 将不具备数据依赖且可并行计算的两个算子进行融合, 从而优化并行度; 最后是同类算子融合, 通过对两个同类算子的融合来减少发射开销或提升访存效率. 通过这些优化措施, 本研究团队实现了跨领域的高效数据流计算生态系统, 为多种计算场景下的性能提升提供了有力支持. 具体而言, 在 DFU 架构上, 本研究团队成功实现了图算子与神经网络算子的融合, 有效支撑了图神经网络 (graphic neural network, GNN) 应用的发展. 同时, 本研究团队在神经网络的关键模块中完成了自注意力模块、全连接模块以及向量检索算子的深度融合, 显著降低了大模型应用对计算带宽的需求, 为高效的大模型计算提供了保障. 此外, 在实时信号处理和检测领域, 本研究团队初步实现了天文大数据处理与合成孔径雷达 (synthetic aperture radar, SAR) 数据处理的原型应用, 为实时处理场景中的高效数据流处理打下了坚实的基础.

5　实验评估

为了验证所提出方法的性能, 实验在如表 4 所示的多种硬件平台上进行. CPU 对比平台采用英特尔 i9-12900HX 处理器, 配置为 16 核心, 主频 5.0 GHz, 计算能力 550 GFLOPS, 功耗 157 W, 内存 64 GB, 运行 Windows 11 操作系统. 异构数据流处理器 (DFU) 平台为自主设计的芯片, 配置为 16 核心, 主频 1.3 GHz, 计算能力 5.995 TFLOPS, 功耗 73.8 W, 内存 64 GB, 并支持 DFU 运行时系统与 DFU 编译器工具链. 由于该芯片尚未投片生产, 实验在基于 FPGA 的 DFU 仿真验证平台上进行, 该平台基于 Synopsys ZEBU Server4, 由 7 块 ZS4 模块、96 块 Xilinx VU440 FPGA 组成, 总计包含约 480 亿门电路, 支持 VCS 编译器.

实验所用的计算负载为合成孔径雷达 (SAR) 处理算法, 输入数据为尺寸为 8K×8K 的雷达回波原始数据, 每个像素点以浮点形式表示, 包括实部和虚部. SAR 算法包含 36 个计算步骤, 覆盖了多种典型计算模式, 包括参数计算、数据类型转换、FFT/IFFT、矩阵向量乘法、插值、矩阵转置和矩阵共轭操作等. 具体处理流程如图 4 所示, 依次包括参数计算、数据读取、RVP 移除、运动补偿、方位补偿、

表 4　硬件实验平台.

Table 4　Hardware experimental platform.

Platform	Model	Hardware	Software
CPU platform	Intel i9-12900HX	16 Cores Frequency: 5.0 GHz GFLOPS: 550 PBP: 157 W DRAM: 64 GB	OS: Windows 11
DFU platform	Dataflow unit (DFU)	16 Cores Frequency 1.3 GHz GFLOPS: 5995 PBP: 73.8 W DRAM: 64 GB	DFU runtime DFU compiler
DFU simulation platform	Synopsys ZEBU server	ZS4 Module×7 Xilinx VU440 FPGA×96 240 M LUTs	VCS compiler

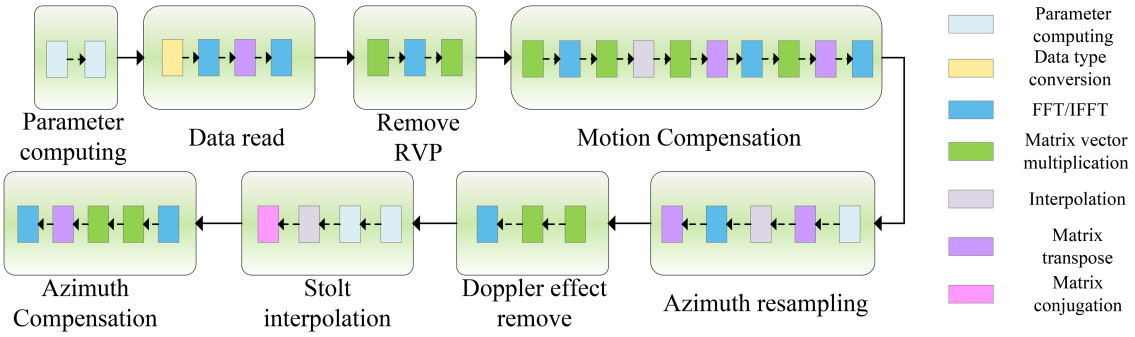

图 4　(网络版彩图) 实验评估所采用的合成孔径雷达 (SAR) 处理算法.

Figure 4　(Color online) Synthetic aperture radar (SAR) processing algorithms used in experimental evaluation.

斯托尔特插值、多普勒效应消除和方位重采样等步骤. 这些步骤组成一个复杂的数据流处理过程, 具有显著的计算密集型特征和多种算法模式的组合, 为评估硬件平台的计算性能和资源利用效率提供了丰富的测试场景.

通过实验对比发现, DFU 异构数据流处理器理论峰值性能为 5995 GFLOPS, 16 核经典控制流 CPU Intel i9-12900HX 理论峰值性能为 550 GFLOPS, DFU 异构数据流处理器理论峰值性能较 16 核经典控制流 CPU Intel i9-12900HX 理论峰值性能提升 10.9 倍, 在 DFU 仿真加速器平台上运行 SAR 算法的时间为 1.168 s, 使用 16 核经典控制流 CPU Intel i9-12900HX 运行相同 SAR 算法的时间为 15.667 s, DFU 异构数据流处理器与 16 核经典控制流 CPU Intel i9-12900HX 相比运行时间提升 13.4 倍. DFU 异构数据流处理器的峰值能效为 81.233 GFLOPS/W, 16 核经典控制流 CPU Intel i9-12900HX 峰值能效为 3.503 GFLOPS/W, DFU 异构数据流处理器与 16 核经典控制流 CPU Intel i9-12900HX 相比, 峰值能效提升 23.2 倍.

6　结论

综上所述, 数据流架构在高性能与高效能计算方面展现出巨大潜力, 为未来计算系统在应对多领

域融合负载时提供了关键的技术支撑. 通过本研究团队对并发代码块数据流抽象机的创新设计及其在多个领域的应用示范, 数据流架构已初步验证了其在图神经网络、大模型计算、实时信号处理等复杂任务中的优越性. 展望未来, 随着数据流芯片与软件生态的进一步完善, 数据流架构有望成为突破传统通用处理器和领域专用处理器性能瓶颈的核心架构之一. 它不仅能够适应各类计算任务的多样化需求, 还能以更低的功耗和更高的灵活性推动人工智能、科学计算、物联网、大数据分析等前沿领域的应用发展. 未来的数据流架构将致力于提供更广泛的兼容性和易用性, 实现跨领域、跨平台的高效计算生态, 为新一代计算系统的研发和应用奠定坚实基础.

参考文献

1 Hameed R, Qadeer W, Wachs M, et al. Understanding sources of inefficiency in general-purpose chips. In: Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA'10), 2010

2 Leng J, Hetherington T, ElTantawy A, et al. GPUWattch: enabling energy optimizations in GPGPUs. In: Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA'13), 2013

3 Jouppi P N, 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'17), 2017

4 NVIDIA. NVIDIA tensor cores unprecedented acceleration for generative AI. 2025. https://www.nvidia.com/en-in/data-center/tensor-cores/

5 TensorFlow. TensorFlow: large-scale machine learning on heterogeneous distributed systems. https://github.com/tensorflow/tensorflow

6 Ansel J, Yang E, He H, et al. PyTorch 2: faster machine learning through dynamic Python bytecode transformation and graph compilation. In: Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'24), 2024

7 Chen T, Moreau T, Jiang Z, et al. TVM: an automated end-to-end optimizing compiler for deep learning. In: Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation (OSDI'18), 2018

8 SambaNova. RDU: the GPU alternative. https://sambanova.ai/technology/sn40l-rdu-ai-chip

9 Burger D, Keckler S W, McKinley K S, et al. Scaling to the end of silicon with EDGE architectures. IEEE Trans Comput, 2004, 37: 44–55

10 Swanson S, Michelson K, Schwerin A, et al. WaveScalar. In: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture (MICRO'36), 2003

11 Allan V H, Jones R B, Lee R M, et al. Software pipelining. ACM Comput Surv, 1995, 27: 367–432

12 Bauer M, Treichler S, Aiken A. Singe: leveraging warp specialization for high performance on GPUs. SIGPLAN Not, 2014, 49: 119–130

13 Shah J, Bikshandi G, Zhang Y, et al. FlashAttention-3: fast and accurate attention with asynchrony and low-precision, 2024. arXiv:2407.08608

14 Groq. Groq® LPU™ inference engine leads in first independent LLM benchmark. https://groq.com/news_press/groq-lpu-inference-engine-leads-in-first-independent-llm-benchmark/

15 Suettlerlein J, Zuckerman S, Gao G R. An implementation of the codelet model. In: Proceedings of the 19th International Conference on Parallel Processing (Euro-Par'13), 2013

16 Zuckerman S, Suetterlein J, Knauerhase R, et al. Using a "codelet" program execution model for exascale machines: position paper. In: Proceedings of the 1st International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era, 2011

17 Kwak J S, Yoon M K, Jeong I, et al. INTERPRET: inter-warp register reuse for GPU tensor core. In: Proceedings of the 32nd International Conference on Parallel Architectures and Compilation Techniques (PACT), Vienna, 2023

18 PyTorch. Deep dive on CUTLASS ping-pong GEMM kernel. 2024. https://pytorch.org/blog/cutlass-ping-pong-gemm-kernel/

Dataflow microprocessor: development, trends, and challenges

Jingwen LENG[1], Minyi GUO[1*], Deze ZENG[2], Wenbin JIANG[3], Xiaochun YE[4], Huaxi CHEN[5] & Wenming LI[4]

1. *School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China*
2. *School of Computer Science, China University of Geosciences (Wuhan), Wuhan 430074, China*
3. *School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China*
4. *State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China*
5. *Intelligent Computing Platform Center, Zhejiang Lab, Hangzhou 311121, China*

* Corresponding author. E-mail: guo-my@cs.sjtu.edu.cn

Abstract This paper explores the potential and trends of novel dataflow architectures in multi-domain integrated computing. Traditional von Neumann and domain-specific architectures struggle to meet the high performance and flexibility demands of emerging technologies like artificial intelligence, graph computing, and big data. We review current dataflow chip design methods, discussing their implementations based on specialization vs. generalization and execution granularity. Based on this, we propose a dataflow abstract machine model using concurrent code blocks, with a complete instruction set and microarchitecture. This model achieves unified intermediate representation across domains and integrates multiple operator fusion strategies, enhancing efficiency in tasks such as graph neural networks, large model computations, and real-time signal processing. Experimental results show that our processor outperforms existing general-purpose architectures in performance and power consumption. We conclude by highlighting the broad application prospects of dataflow architectures in future computing systems and their significant impact on efficient computing.

Keywords multi-domain integrated computing, dataflow architecture, abstract machine model