计 算 机 研 究 与 发 展
Journal of Computer Research and Development

https://crad.ict.ac.cn
62(5): 1091−1107, 2025

Neptune: 一种通用网络处理器微结构模拟和性能仿真框架

林涵越[1,2] 吴婧雅[1] 卢文岩[1] 钟浪辉[3] 鄢贵海[1]

[1](处理器芯片全国重点实验室(中国科学院计算技术研究所) 北京 100190)

[2](中国科学院大学计算机科学与技术学院 北京 100049)

[3](上交所技术有限责任公司 上海 200131)

(liuhanyue21b@mails.ict.ac.cn)

Neptune: A Framework for Generic Network Processor Microarchitecture Modeling and Performance Simulation

Lin Hanyue[1,2], Wu Jingya[1], Lu Wenyan[1], Zhong Langhui[3], and Yan Guihai[1]

[1] (*State Key Lab of Processors* (*Institute of Computing Technology*, *Chinese Academy of Sciences*), *Beijing* 100190)

[2] (*School of Computer Science and Technology*, *University of Chinese Academy of Sciences*, *Beijing* 100049)

[3] (*Shanghai Stock Exchange Technology Co.*, *Ltd.*, *Shanghai* 200131)

Abstract Network packet processing is a fundamental function of network devices, involving tasks such as packet modification, checksum and Hash computation, mirroring, filtering, and packet metering. As a domain-specific processor, network processor (NP) can provide line-rate performance and programmability for network packet processing. However, due to different design requirement, architecture of NP differs, including single-phase NP and multi-phase NP, posing challenges for NP designers. Existing simulation methods mainly target single NP or single architecture and are not available to explore both of the architectures. We propose Neptune, an analyzing framework for generic network processor microarchitecture modeling and performance simulation. Based on detailed analysis, Neptune adopts multi-phase NP architecture as the hardware model while providing ability to simulate single-phase architecture. Besides, Neptune employs event list mechanism and inter-core queues to support simulation of different data paths and various scheduling strategies in multi-phase NP. Furthermore, Neptune utilizes bulk synchronous parallel graph computing mechanism and takes advantage of both event-driven and time-driven simulation, ensuring accuracy and efficiency. Our experiment shows that Neptune achieves over 95% accuracy in simulating both of the architectures and simulates network processors at a performance of 3.31 MIPS, achieving an order of magnitude improvement over PFPSim. We illustrate the universality and capability of the Neptune simulation framework through three specific cases. Firstly, we evaluate multi-phase and single-phase NP, showing that single-phase NP can achieve up to a 1.167 times performance improvement. Secondly, we optimize the packet parsing module using a programmable pipeline and analyze its performance differences. Finally, we use Neptune to test the performance of the network packet processing engine under different thread counts, providing insights for software and hardware multi-threading optimization.

Key words network packet processing; network processor; programmable data plane; domain-specific processor; simulator

收稿日期: 2024-02-02; **修回日期**: 2024-09-03

基金项目: 国家自然科学基金项目(62002340, 61872336, 61572470); 中国科学院青促会基金项目(Y201923)

This work was supported by the National Natural Science Foundation of China (62002340, 61872336, 61572470) and the Program of the Youth Innovation Promotion Association, CAS (Y201923).

通信作者: 鄢贵海(yan@ict.ac.cn)

摘 要 网络包处理是网络设备的基本功能,涉及报文修改、校验和与哈希计算、数据包镜像或过滤、统计限速等多项任务.作为网络包处理的重要部件,网络处理器(network processor, NP)基于处理器结构,为网络设备提供线速的性能和充分的可编程能力,但其架构多样,可分为单段式架构和多段式架构,现有模拟方法无法同时对二者性能进行模拟仿真.因此,提出一种通用网络处理器的结构模拟和性能仿真框架Neptune,采用多段式架构作为硬件抽象,使用事件链表、核间队列结构为数据通路和多段式架构模拟提供保障,同时满足单段式架构模拟需求.另外,借助同步图计算模式进行准确的并行模拟,并采用混合事件与时间驱动方法保障模拟高效性.实际测试中,Neptune以95%以上准确率支持2种架构的模拟,并以3.31MIPS的性能对网络处理器进行模拟,相较PFPSim取得1个数量级的性能提升.最后,展示了3个运用该框架进行网络处理器优化分析的应用案例.

关键词 网络包处理;网络处理器;可编程数据面;专用处理器;模拟器

中图法分类号 TP302

DOI: 10.7544/issn1000-1239.202440084 **CSTR:** 32373.14.issn1000-1239.202440084

网络包处理是网络设备的基本功能,涉及报文修改、校验和与哈希计算、数据包镜像或过滤、统计限速等多项任务.然而,网络包处理业务的快速迭代与网络设备的开发速度不匹配,网络设备需要一定的可编程能力,以获得更长的使用周期.与此同时,互联网带宽的不断增加也对网络包处理的性能带来挑战.网络处理器(network processor, NP)基于处理器结构提供了丰富的网络可编程能力,并通过众核架构和专用指令集加速网络数据包处理,获得线速的网络处理能力[1].它在路由器[2]、智能网卡[3-4]和数据处理单元(data processing unit, DPU)[5]等网络设备中,都发挥着重要作用.

然而,由于网络包处理需求多样,网络处理器中并不存在绝对的最优架构.根据调研,我们可将众核网络处理器架构分为单段式架构和多段式架构.单段式架构中,每个网络处理器核负责对网络数据包的全部处理;而多段式架构中,网络数据包处理被拆分为多个子任务,不同的处理器核负责不同的部分,通过流经多个网络处理器核完成处理.这2种网络处理器架构各有优劣,网络处理器的设计人员希望能基于网络处理器设计目标,在设计早期便与网络包处理业务人员展开架构探索,选择合适的网络处理器架构.

为了实现这一目标,网络处理器的模拟与性能分析发挥着关键的作用.作为体系结构研究中的重要工具,模拟器使用高层语言对底层硬件进行抽象,能在设计早期就对硬件行为进行精确模拟,以此对芯片性能进行评估和优化.在网络处理器设计中,模拟器能够以更快的速度探索芯片设计空间和加速芯片设计,为网络处理器研究人员提供关于2种网络处理器架构的关键理解.然而,由于网络处理器的专用性,已有的网络处理器模拟器往往只能模拟特定的网络处理器产品[6],或只能模拟特定处理架构[7],难以对网络处理器进行更全面的架构探索.我们无法基于现有的模拟框架,对网络处理器中的2种通用计算架构进行差异分析和相关优化.

因此,本文提出了Neptune,一种通用网络处理器的结构模拟和性能仿真分析框架,以满足设计人员对不同网络处理器架构的性能差异分析需求.相较于其他模拟方案,Neptune采用多段式架构作为网络处理器硬件抽象模型,并能以此兼容单段式架构的模拟.在此之上,Neptune引入离散模拟中的事件链表机制为数据传输进行模拟,并通过硬件核间队列的模拟,为多段式调度提供架构探索空间.此外,Neptune借鉴已有模拟器的周期同步方法,提出以同步图模拟和混合事件驱动的方式提高模拟性能.Neptune能集成适配P4测试环境,并在测试中以超过95%的准确率对不同架构在真实应用场景中的处理性能进行模拟,达到3.3 MIPS的模拟速度.最后,我们展示了如何使用Neptune对单段式处理架构和多段式处理架构进行全面的性能分析.

本文的主要贡献包括3个方面:

1)调研并总结了现有网络处理器中的架构方案,将网络处理器架构分为单段式架构、多段式架构,并提出通用网络处理器结构模拟和性能仿真框架Neptune.与过去网络处理器模拟器研究不同,Neptune基于多段式架构硬件模型,同时为单段式架构、多段式架构提供支持,并通过事件链表、核间队列架构为数据通路和多段式架构模拟提供保障.

2)Neptune根据网络处理器模拟特点,采用同步

图模拟和混合事件与时间驱动的方法以保障模拟器准确性和高效性，并能与现有 P4 框架集成. Neptune 能支持不同的网络处理器架构的模拟，在真实应用中延时偏差不超过 5%，性能相比 PFPSim[7] 取得数量级的提升.

3）通过 3 个具体案例展示 Neptune 在架构探索中的应用潜力. Neptune 可以评估不同段式处理器的数据包处理性能差异，发现单段式架构相比多段式架构能取得性能提升并为多段式架构提供集群配置参考. 介绍了如何通过 Neptune 对异构多段式架构进行性能评估，及如何对网络处理器核内包处理线程数带来的影响进行评估.

1 研究背景和相关工作

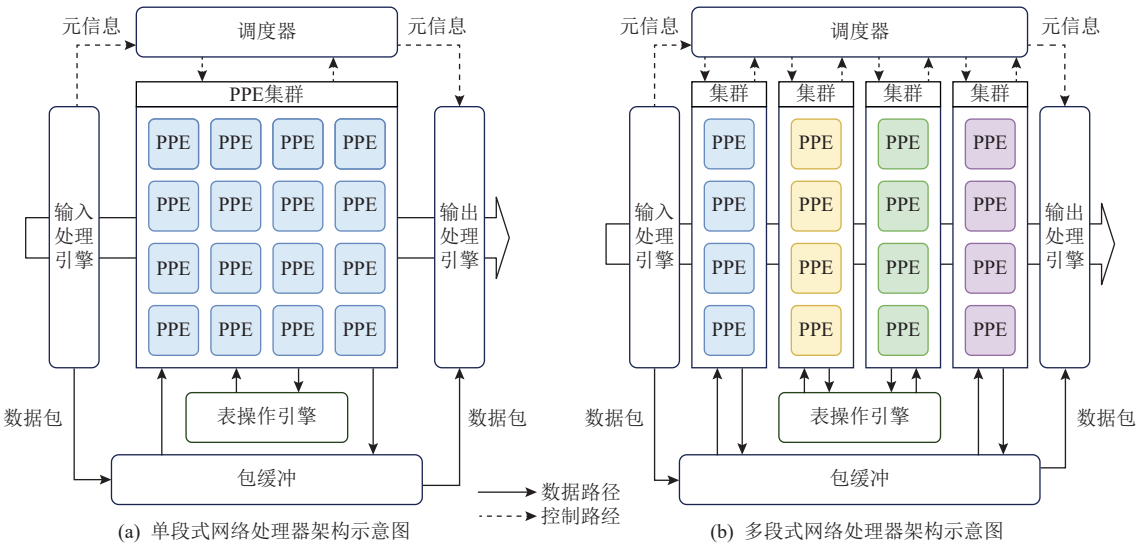
1.1 单段式与多段式网络处理器

随着网络带宽的快速增加和包处理任务的快速迭代，现代网络对于高度灵活且高性能的网络包处理平面的需求日益增长. 当前，网络数据平面可编程方案分为可编程流水线方案和网络处理器方案. 可编程流水线主要基于可重构匹配表（reconfigurable match table, RMT）[8] 方案，能兼顾流水线性能的同时，保障可编程能力. 然而，它的包处理能力受流水线级数限制，且难以支持需要循环、函数调用的处理操作.

与之相比，网络处理器具有和通用处理器相似的处理器结构，基于软件方式对网络进行处理，具有更大的灵活性，更适合于网内计算的应用[2]. 通过设计专用的指令集、存储和计算单元，并利用众核架构的并行性，网络处理器能提供远超通用处理器的包处理性能. 除了常规包处理功能外，它还能承担起协议卸载[9]、无服务计算[10]、包调度[11] 和深度包检测[12] 等多种复杂包处理任务. 然而，多样的处理场景也导致不同网络处理器的架构不同. 通过调研和总结，我们根据网络数据包在 NP 中的处理步骤，将众核网络处理器分为单段式和多段式 2 种处理架构.

单段式架构通常指由 1 个网络处理器核负责完整网络包处理任务的网络处理器架构，这一概念与 RTC（run-to-completion）相近. 单段式架构下的网络处理器模型如图 1（a）所示，其中，同构的包处理引擎（packet processing engine, PPE）负责执行完整的包处理业务. 数据包在经过输入处理引擎预处理后，就被调度到单个 PPE 中，完成全部包处理. 单段式架构编程简单，不需要额外的数据通信. 在工业界，思科公司的 Silicon One 系列产品[13] 即采用单段式处理架构，瞻博公司的 Trio 系列网络处理器[2] 也可被划为此列. 在学术研究中，同样也有一些单段式架构的 PPE 或网络处理器设计[14-15]，例如，dRMT 架构将 P4 的 RMT 可编程流水线架构转换为单段式架构，为 P4 语言在众核网络处理器中的适配提供了参考. 相比于可编程流水线，dRMT 只需要更少的处理器即可完成相同的处理需求[16].

与单段式架构不同，多段式架构中，网络处理器被拆分成不同的 PPE 集群（cluster），包处理任务由多个集群中的 PPE 协同完成，图 1（b）所示. 多段式架构对包处理任务进行分解，与 multipass 相近.在网络处理陷入突发拥塞时，多段式架构能为小包的服务质



(a) 单段式网络处理器架构示意图　(b) 多段式网络处理器架构示意图

Fig. 1 Illustration of different architectures of network processors

图 1 不同网络处理器架构示意图

量(quality of service, QoS)提供更好的保障. 同时, 多段式架构也给芯片优化提供了空间, 可以根据不同集群的功能不同, 对其内部的处理单元进行调整, 对部分 PPE 集群进行优化. 在多段式架构中, PPE 可以是异构的, 甚至可以用专用硬件进行替换. 但由于数据包需要多次处理, 多段式架构的编程往往更为复杂, 不同段处理间也会带来开销. 在工业界, Intel IPU 芯片[17] 和 Netronome 智能网卡[18] 的网络包处理模块都采用了多段式网络处理器架构. 而在学术界, Yazdinejad 等人[19] 提出的 NPPP 网络处理器架构也采用了多段式架构的处理模式. ChipletNP 基于 Chiplet 设计思想, 通过多个敏捷交换芯粒异构互联, 协同完成网络包处理任务[20], 同样是一种多段式网络处理器架构.

虽然单段式架构和多段式架构都在网络处理器中得到了相应的应用, 然而, 目前并没有已知的研究对这 2 种处理架构进行具体差异分析. 对于网络处理器开发人员而言, 如何根据网络场景选择适合的网络处理器架构仍需要深入探索.

1.2 网络处理器结构模拟与性能仿真

在芯片设计中, 设计人员往往希望能在设计初期就对硬件架构进行设计空间和性能探索. 而在网络处理器设计中, 除专用的指令集和专用的包处理引擎架构设计外, 单段式架构和多段式架构的选择也存在架构探索空间.

网络处理器的建模与性能评估存在不同方法, 包括数学建模、硬件分析、仿真模拟等. 数学建模方法主要基于排队论开展相关研究. 文献 [21] 通过排队论的方法探究了控制-数据平面结合的主-从网络处理器架构的性能表现, 并以此对网络处理器的性能进行建模和评估, 不过这一研究对于目前的众核网络处理器架构并不适用. Keslassy 等人[22] 基于排队论, 探索了 multipass 网络处理器架构下网络处理器的缓冲区和调度策略的选择. Zolfaghari 等人[23] 则使用硬件分析方法进行评估, 比较了使用处理器核和流水线结构进行包解析带来的性能差异.

然而, 理论计算通常难以对硬件的真实情况进行反映, 而基于硬件原型的分析又需要漫长的开发周期. 因此, 在早期设计中, 基于模拟器的性能模型是一种极为重要的性能评估方法. 网络模拟器如 OMNeT++[24], NS4[25] 等提供了对网络互联和拓扑的模拟. Dons 模拟器[26] 对网络模拟器底层软件架构进行了优化, 显著提升了网络模拟器的性能. 不过这些模拟器都并没有对硬件架构进行模拟. 众核模拟器, 如 McSimA+[27] 提供了对复杂的 x86 处理器核的多核模拟; HORNET[28] 则提供了基于时钟级别的虫孔路由 NoC 架构的多核模拟; Gem5-X[29] 基于 Gem5 模拟器, 提出了一种异构众核模拟的探索调优平台. 但网络处理器大多并不采用 x86 等复杂指令集, 而 Gem5-X 也还没有被应用于网络处理器的设计中. 还有一些工作, 如 TONIC[30], 对其专用的网络处理部件提供了精确的模拟, 但这与本文所探讨的网络处理器不直接相关.

令人遗憾的是, 专为网络处理器硬件架构设计探索的模拟器实现并不多见. 在学术界, NePSim[6] 作为较早的工作, 以 Intel IXP-1200 网络处理器为基础架构, 进行了准确的时钟和功耗模拟, 然而, 这也限制其应用空间, 使得其仅局限于单一的处理器架构中. 此外, Wagner 等人[31] 提出了一种对 RTC 架构的网络处理器的快速编译、模拟、调试的套件. Koohi 等人[32] 则提出 SNP 模拟器, 对同步多线程(SMT)模式的网络处理器给出模拟. PFPSim[7] 基于 SystemC 实现了支持 P4[33] 语言的网络处理器模拟方法, 使得其能够比较网络处理器与可编程流水线的处理性能. 但这些工作所模拟的网络处理器架构均为 RTC 架构, 属于单段式处理的范畴, 并没有对多段式计算架构进行模拟. SimBricks[34] 提供了端到端的网络系统模拟方案, 能够用以评估智能网卡的性能, 但其提供的 Corundum 网卡仿真模型并不属于网络处理器, 需要基于硬件 RTL 代码进行网络处理器架构探索, 在设计空间探索时作用有限. 而在工业界, 虽然 Netronome 公司的 Programmable Studio 6.0 中提供了基于其产品的多段式架构模拟器[35], 但由于其仅限于单一产品, 这一模拟器同样不具有早期网络处理器设计空间探索的价值. 因此, 为了对网络处理器的 2 种计算框架进行探索, 我们需要设计新的网络处理器模拟器, 使其在提供良好的模拟性能和准确性的同时, 对单段式和多段式架构进行支持, 为设计人员提供更全面的设计空间探索.

1.3 模拟器设计需求

在体系结构研究和芯片系统设计中, 模拟器发挥着极为重要的作用. 它通过高级语言, 对底层硬件进行抽象, 对其性能进行模拟. 模拟器既能为上层软件开发人员提供一个早期验证平台, 使其在无具体硬件的情况下也能进行软件开发和功能迭代; 又能为底层硬件设计人员提供硬件功能验证, 更快发现潜在的设计缺陷来优化架构, 从而缩短产品开发周期. 而为了实现上述功能, 网络处理器的模拟器需要

满足 3 个功能需求：

1)通用性. 网络处理器作为专用于网络处理的器件, 其架构、指令呈现出多样的特点. 具体而言, 网络处理器模拟器需要能支持单段式和多段式架构, 并通过松耦合提供不同模块的可修改、可扩展性. 这样, 模拟器的应用价值才能得以保障. 不过, 不能因此而使得模拟器过于抽象和简化, 使其难以被应用于具体的模拟场景中.

2)准确性. 在模拟过程中, 模拟器必须准确模拟硬件中的执行顺序, 以此保证模拟结果与硬件结果基本一致. 过大的模拟偏差将导致模拟器无法在实际场景中得到应用.

3)高效性. 在保证准确性的基础上, 模拟器的性能也至关重要. 过长的模拟时间将使得大型的模拟变得不现实. 因此, 模拟器需要在准确的前提下, 消除冗余的计算, 以此保证实现的高效性.

2 Neptune: 网络处理器模拟框架设计方法

在本节中, 我们将介绍所提出的网络处理器模拟系统 Neptune 的具体实现方法. 基于模拟器设计需求, 下文将介绍 Neptune 系统的设计原则, 并对 Neptune 系统中的硬件模型、模拟方式等进行介绍.

2.1 设计原则

为符合模拟器设计需求, 满足相应功能特性, Neptune 提出并在设计中遵循了以下 3 个设计原则: 以多段式架构为框架、同步图式模拟、混合事件与时间驱动.

1)以多段式架构为框架, 满足模拟器通用性和准确性需求. 根据调研与分析, Neptune 采用多段式架构作为网络处理器架构模拟的框架. 通过修改处理段数, 调整集群规模, 多段式架构同样可以适配单段式架构, 让 Neptune 对多种网络处理器架构进行模拟, 满足模拟器设计的通用性需求. Neptune 还将网络处理器架构模块化, 将之拆分为包接收/发送模块、包处理引擎、数据通路、核间调度模块等. 其中, 引入离散事件模拟中[36]的事件链表结构对 2 种架构的数据通路进行统一模拟, 使用硬件中常用的核间队列结构对多段式核间调度进行支持. 解耦化的设计使得模拟器更加通用, 对硬件架构的准确模拟也满足准确性需求.

2)同步图式模拟, 满足模拟器准确性和高效性需求. Neptune 将每个模块看作 1 个图节点, 以时钟为迭代周期, 将网络处理器的模拟看作图计算. 图计算方法分为同步图计算和异步图计算 2 种. 虽然异步图计算框架能取得更好的性能, 但难以通过简单的机制保障模拟的准确性. 而同步图模拟可以通过周期内并行, 保障模拟的高效性. 因此, Neptune 借鉴以往时钟同步模拟器[28]的设计, 并采用同步图计算的方式, 以时钟周期为单位进行同步, 周期内各模块并行模拟. 通过这种方式, 保证了准确性和高效性.

3)混合事件与时间驱动, 满足模拟器高效性需求. 通常, 模拟器的驱动方式包括事件驱动和时间驱动. 时间驱动中, 时钟周期逐步累加, 轮询执行每个触发的事件; 事件驱动则相反, 选择事件队列中下一个可执行的事件, 将时钟切换到事件的触发时钟后执行这一事件, 在事件较少时, 能取得更高的执行效率. Neptune 根据网络处理器的特点, 混合了时间驱动和事件驱动 2 种框架, 整体上以时间驱动的方式对其进行模拟, 当没有包处理过程执行时, Neptune 通过事件驱动的方式, 将模拟时钟切换至下一个包输入时间, 保障模拟的高效性.

基于以上设计原则, 我们设计并实现了 Neptune 结构建模和性能仿真框架. 这一框架能准确通用地模拟网络处理器中的单段式架构和多段式架构, 为网络处理器的设计空间探索和跨层优化提供支持.

2.2 包处理理论和 Neptune 硬件模型

正如 1.2 节所指出, 目前学术界所提出的网络处理器模拟器多基于单段式架构进行网络处理器建模, 而无法对多段式架构进行支持. 工业界虽然有多段式架构的网络处理器模拟器, 但仅能对单个产品进行模拟, 不具备设计空间探索的能力. 为了保证模拟器的通用性, 需要模拟器能对网络处理器中的单段式架构和多段式架构都进行模拟. 而通过理论分析和架构分析, 我们不需要设计 2 套硬件模型分别构建模拟框架, 只需对多段式架构进行模拟即可对单段式架构进行支持.

从包处理理论的角度, 对于输入的数据包 p 和网络处理器状态 s, 网络处理器对数据包进行处理后, 将输出 0 到多个数据包. 记数据包集合为 P, 网络处理器状态集合为 S, 我们可将网络数据包处理过程抽象为包处理函数 $F(p, s)$, 表示为

$$F(p,s)=(p_1p_2\cdots p_n,s'):P\times S\rightarrow P^*\times S.$$

对于单段式网络处理架构而言, 每个 PPE 执行完整的函数处理 $F(p, s)$. 但对于多段式函数处理而言, 可将功能不同的集群中看作 n 个函数组 $f_1, f_2, \cdots, f_n$, 其集合记作 FS. 由此, 可将多段式包处理函数看作不同集群处理函数的复合, 写作

$$F(p,s)=f_{i_1}(f_{i_2}(\cdots f_{i_k}(p,s)\cdots)),\ f_{i_1},f_{i_2},\cdots,f_{i_k}\in FS.$$

不难看出，当 n=1 或 k=1 时，多段式架构的包处理函数实际上就是单段式架构中集群的包处理函数 $F(p,s)$. 因此，虽然单段式架构无法兼容多段式模拟，但如果从多段式的角度对网络处理器进行建模，将能同时满足 2 类架构的建模需求.

事实上，从架构分析的角度，当多段式处理架构中只存在单个集群时以及每个 PPE 都执行完整包处理过程时，这种多段式架构实际上就是一种单段式处理架构. 不难看出，多段式架构具有对单段式架构的兼容能力. 采用多段式架构作为硬件模型时，便能对 2 种架构进行支持.

因此，与过去的网络处理器模拟框架不同，Neptune 采用多段式架构作为网络处理器硬件模型进行模拟，其示意图如图 2 所示. 在模拟器中，具有相同处理指令的 PPE 构成集群，通过调度的方式使得一个数据包流经多个 PPE，完成处理并发送. 而通过修改网络处理器的拓扑和调度配置，Neptune 可以同时为单段式架构模拟提供支持. 在具体实现中，Neptune 支持对集群数和集群内的 PPE 个数进行调整，并提供了其他扩展模块，如可编程流水线、协处理加速引擎等，使得网络处理器可以适应广泛的架构优化场景.

在 Neptune 模拟硬件模型中，模拟的关键在于对 PPE、核间和核外总线以及调度的模拟. PPE 是网络处理器中最重要的结构，负责对网络包进行各类处理. 在单段式架构中，PPE 之间不进行直接的数据交互，但在多段式架构中，数据通路分为 2 类，分别是在 PPE 之间进行数据交互的核间通路，以及与外部处理引擎、包缓冲进行数据交互的核外通路，因此，需要对 2 种数据通路均进行支持. 另外，针对多段式的架构中数据包在 PPE 集群之间的传输，Neptune 还需要对多段式架构中核与核之间的调度策略提供基本的模拟支持.

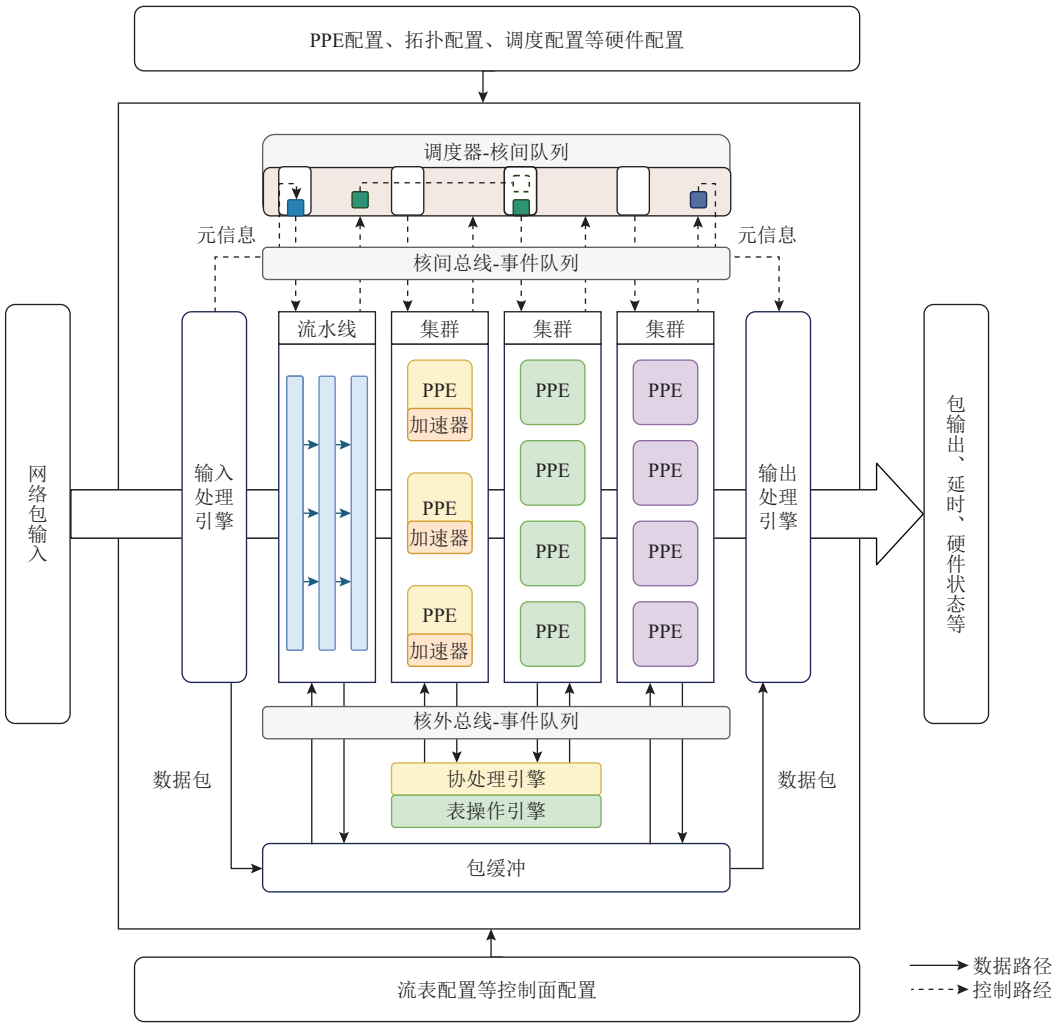


Fig. 2 Illustration of simulated hardware architecture of Neptune

图 2 Neptune 模拟硬件架构示意图

针对上述问题，我们设计了 3 个不同的模块进行模拟. 对于 PPE 的模拟，我们给出一个参考的包处理引擎模型，供设计人员进行调整和适配. 而对于数据通路的模拟，Neptune 引入了事件链表的数据结构，以事件驱动方式对数据通路进行模拟. 而为了实现不同的调度策略，Neptune 引入了和硬件相似的核间队列结构，使其能对不同的调度策略进行支持. 在下文中，我们将对这几个关键模块的模拟进行介绍.

2.3 PPE 模拟

网络处理器的最小功能执行单元为 PPE，这是一种类似 CPU 架构的专用处理器. 由于网络包处理利用指令级并行性的空间较少，因而，多发射对于网络处理的性能提升较少[37]. 大多数 PPE 架构中采用单发射的架构设计，通过增加 PPE 核数，利用众核并行处理的方法提供高吞吐量.

Neptune 提供了一种参考 PPE 架构，如图 3 所示. 这是一个具有 4 级流水的单发射处理器架构，包括取指、译码、执行、写回 4 个流水级. Neptune 采用了软件流水的方法，在同一时钟周期，以倒置顺序对 4 级流水线进行模拟. 在 PPE 内部，具有算术逻辑单元、数据搬移单元、特殊计算单元 3 个执行模块，数据则保存在通用寄存器、特殊寄存器以及核内缓冲区中. 当需要与外部处理引擎进行交互，或进行核间调度时，PPE 会将数据通过核内发送队列发出，加入核间或核外事件队列，并暂停流水线. 事件完成后，通过接收队列告知 PPE 进行后续处理. 值得一提的是，由

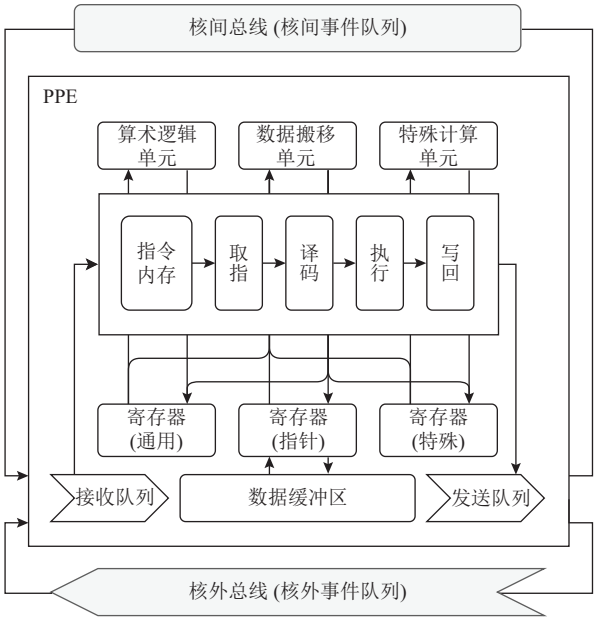


Fig. 3 Illustration of PPE architecture

图 3 PPE 架构示意图

于不同 PPE 之间数据共享的情况较少，我们没有引入高速缓存同步机制，而是通过将包缓冲、表项读入核内缓存并写回，实现相应功能. 对于表项冲突问题，通过核外加锁等机制进行解决.

Neptune 参考的 PPE 架构来源于工业界的设计. 通过对流水线以及运算、存储部件的扩展和修改，Neptune 也能够快速支持其他 PPE 处理器设计的模拟，为后续的架构优化和跨层优化提供支持.

2.4 数据通路模拟

总线是负责 PPE 与外部模块进行数据传输互联的关键，在多段式架构中，总线可分为核间互联和核外互联 2 类. 核间互联总线负责 PPE 到 PPE 之间的元信息传递. 而当涉及流表查询、包缓冲区等操作时，PPE 将通过核外总线，把数据传输至对应模块进行计算，其结果也通过核外总线返回.

Neptune 并不直接对数据通路进行模拟，而是以事件的抽象对 PPE 和外部的通信进行实现. Neptune 引入了离散事件模拟中常用的事件链表结构[36] 以实现单段式架构和多段式架构中统一的数据通信机制，其结构如图 4 所示. 事件链表由事件节点构成，每个事件节点包括事件类型、事件参数、事件触发时间等信息. 当 PPE 发起相应请求时，事件链表将创建新的事件节点，并将其插入总线中. 当触发事件的时间来临时，通过函数 *check_event*() 和 *handle_event*()，事件与外部引擎进行交互.

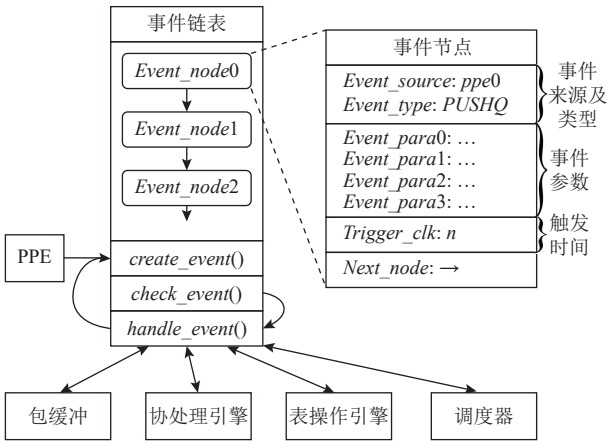


Fig. 4 Illustration of the data structure of event list

图 4 事件链表数据结构示意图

通过以上方式，Neptune 使用一种数据结构对 2 类总线进行实现，以此对单段式、多段式架构进行支持. 如果是核外总线，事件将与包缓冲、协处理引擎、表操作引擎等模块进行交互，如果是多段式架构中的核间总线，则事件与核间调度器和核间队列进行

交互. 而通过调整事件的触发延时, Neptune 可以实现对总线的近似模拟. 触发延时可以来自人为设定的硬件测试延时, 也可以来自其他总线模拟器得出的结果.

2.5 核间调度模拟

在多段式架构中, 数据包与单段式架构不同, 将在不同 PPE 集群间调度. 为了模拟数据包在多个处理引擎集群间的调度过程, Neptune 引入了与硬件中相似的核间队列的结构, 为多段式架构模拟提供额外的支持. 通过对核间队列的配置, 网络处理器设计人员可以自行设计相应的调度器和调度方式, 使其通过收集核间队列的信息, 完成数据包到包处理引擎的调度.

图 5 中展示了 Neptune 中核间队列数据结构的示意图. 该图中, 每个核间队列保存有输入数据包描述符信息, 以队列形式缓冲尚未到来的数据包. 当数据包从输入引擎进入网络处理器进行调度, 或 PPE 发起调度请求时, 它所携带的元信息将作为事件参数进入核间总线的事件队列, 并在触发时间进行处理, 转换为描述符信息, 并加入核间队列的输入队列的队尾.

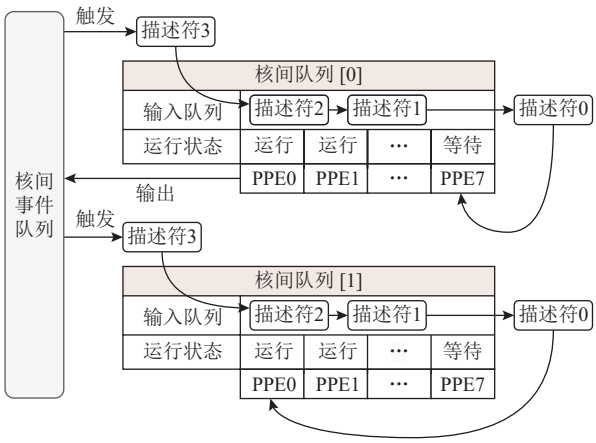


Fig. 5 Illustration of the data structure of core queues

图 5 核间队列数据结构示意图

基于核间队列, 网络处理器设计人员可以探索核间调度的不同情况和策略. 示意图中采用了简单的“先来先服务”(first come first serve, FCFS)的调度方法. 每个核间队列保存了一个 PPE 集群下的所有 PPE 运行状态信息. 当其中一个 PPE 从运行状态切换至等待状态后, 调度器收集到这一信息, 当核间队列内有元素时, 调度器将队头元素弹出, 并推入对应的 PPE, 使其对队头数据包进行处理.

2.6 同步图模拟与混合驱动

基于 2.3~2.5 节所述的模拟模块, Neptune 的顶层模拟框架如图 6 所示. Neptune 将网络处理器的处理流程根据数据通路划分为核间和核外 2 个部分. 核间部分包括输入引擎、输出引擎以及在路径上的不同的 PPE 集群. 核外部分则包括数据包缓冲区及其管理模块、保序模块、表处理模块等的模拟, 这一部分可由设计者进行扩展. Neptune 以集群作为模拟的基本单位, 集群内包含同构、同功能的网络处理部件, 例如 PPE 或可编程流水线, 而不同集群之间则可以是同构或异构形式. 这种模块化的设计既与硬件设计相契合, 又能适应广泛的修改和扩展.

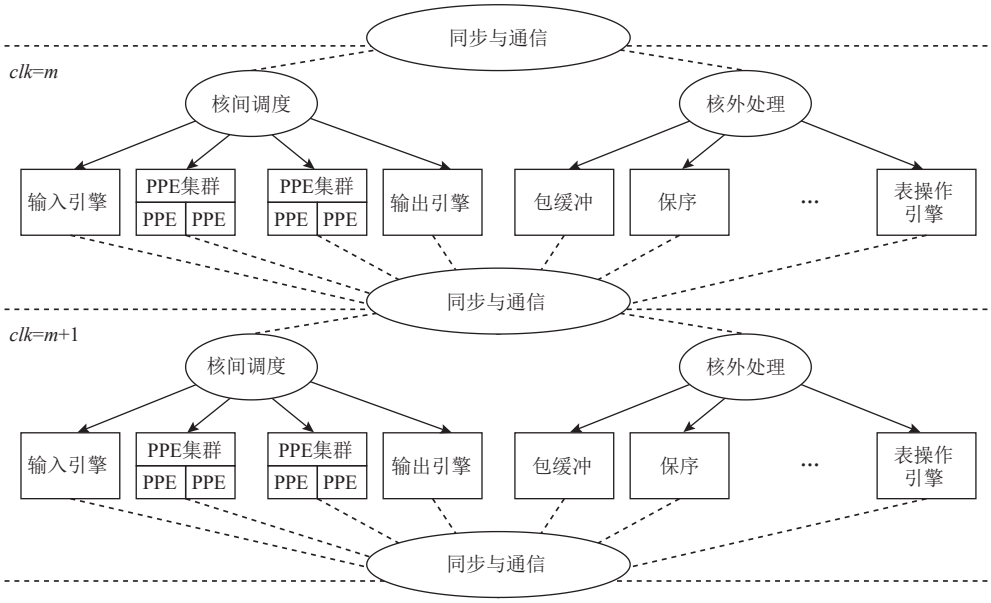


Fig. 6 Illustration of Neptune top-level simulation architecture

图 6 Neptune 顶层模拟架构示意图

如我们在 2.1 节中所提到，Neptune 参考了 HORNET[28] 等多核模拟器的周期同步策略，将网络处理器的模拟看作一种同步图计算过程，将模拟拆分成了模块计算和同步通信部分. 在注入配置后，每个时钟循环时，Neptune 都会检查不同模块是否有需要执行的事件并执行，生成新的执行事件. 而在同步通信阶段中，Neptune 收集相关事件进行处理，并将新事件传递给核间模块和核外模块执行，这种模拟严格按照时间执行，从而保障模拟准确性. 另外，Neptune 模拟框架还使用 OpenMP 以保证程序并行性. 由于不同数据包处理过程之间干扰较少，Neptune 以集群为单位，对网络处理器进行线程划分，通过计算机并行性对模拟过程进行加速.

此外，网络处理器是对网络数据包进行处理的处理器，具有数据包驱动的特点，即在无数据包处理任务时，其状态不会发生改变. 因此当模拟的网络数据流较为稀疏时，纯粹时钟驱动的方法会导致模拟过程中无效的循环操作. 为了避免这一情况，Neptune 引入了混合事件和时钟驱动方法. 如果在当前时刻数据包处理已全部完成，Neptune 将会根据事件驱动的思想，将时钟切换到下一个数据包输入时间，消除冗余计算，减少无意义的轮询开销，保证模拟的高效性.

3 集成与测试方法

模拟网络处理器的行为需要几组关键的预输入，包括网络数据包输入的模拟、配置信息的模拟、PPE 指令的模拟等. 在本节，我们将介绍 Neptune 如何运用于测试中. 具体而言，Neptune 提供了一套原生的集成测试框架；另外也可以结合现有的网络测试框架，对支持 P4 的网络处理器进行模拟和交叉验证.

3.1 Neptune 原生测试框架

Neptune 自身自带一套离散事件式模拟配置方法. 对于网络数据的输入，以（输入文件，输入时间）的二元组形式形成文件，作为 Neptune 的输入. 同时，配置信息也以文件的形式进行加载.

对于 PPE 指令的模拟需要与具体指令集相结合. 在参考实现中，Neptune 支持了一套网络处理器专用精简指令集，包括算术、数据搬移、流表查询修改等指令. 使用者通过设定不同集群所使用的指令集，以调节不同 PPE 的配置.

3.2 Neptune 在 P4 环境下的集成与测试

虽然 Neptune 提供了一套基于离散事件和指令微码的集成测试方法. 但在实际场景中，业务人员除了通过事件模拟测试性能外，还需要在实际场景中验证其功能正确性. 另外，繁冗的指令微码编写也对业务人员构成挑战.

对此，Neptune 提供了一套基于 P4 环境的集成测试方法，该方法基于 P4 行为模型 BMv2（behavioral model v2）设计. BMv2 是一种基于 P4 语言，允许用户自定义网络交换机的行为和数据平面处理逻辑的 P4 应用模拟和测试平台①. 它能以灵活和可编程的 P4 语言对报文的处理和转发行为进行配置，将 P4 程序编译为高效的底层数据平面描述，以实现高性能的数据包处理和转发，被广泛用于 P4 网络交换机等案例中.

Neptune 可以在继承 BMv2 的外层框架的同时，通过对 BMv2 数据面的修改，实现在 BMv2 框架中的集成，如图 7 所示. P4 的代码在经过 P4 编译器后，生成控制面配置和数据面配置. 而通过修改编译器，可以使得 P4 提供对 BMv2 数据面的支持外，还能额外对网络处理器的相关配置进行支持. 在数据包进入后，将同时流经配置好的 BMv2 和 Neptune 框架中，并可对输出进行比较.

4 验证及性能评估

本节我们将对 Neptune 模拟框架相应特性进行验证与评估.

4.1 Neptune 通用性检验

为了验证 Neptune 系统的通用性，我们基于 Neptune，选取了中科驭数（北京）科技有限公司在设计其 DPU 产品中的网络处理器时曾采用的 2 种具有代表性的单段式、多段式架构网络处理器设计方案作为样例，展示了该方法的通用性.

Neptune 生成的单段式和多段式架构相关参数如表 1 所示. 所模拟架构具有相同的时钟频率和包处理引擎数，多段式架构中的 PPE 仍是同构的. 通过修改集群数为 1，Neptune 此时实现了对单段式架构的模拟. 而对于多段式架构，每 8 个 PPE 构成 1 个 PPE 集群，16 个集群构成完整的网络处理器核心结构. 网络处理器被分为 3 段，对应包解析、匹配执行和再封装功能，各段分别有 4, 8, 4 个同构集群进行处理. 由于

① https://github.com/p4lang/behavioral-model

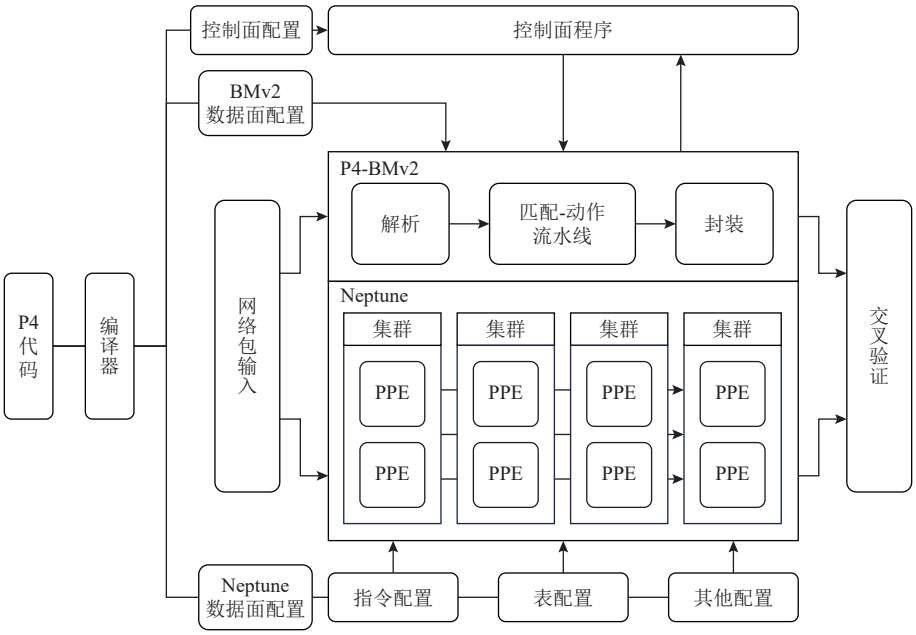


Fig. 7 Illustration of integrating BMv2 with Neptune

图 7 BMv2 集成 Neptune 示意图

Table 1 Configuration of Parameters Required for Function Verification of Neptune

表 1 Neptune 功能验证需要的参数配置

参数类型	参数	数值
通用参数	时钟频率/MHz	800
	PPE 内线程数	1
	核内内存大小/B	1 024
	包缓冲大小/MB	1
单段式架构参数	PPE 总数	128
	集群数	1
	每集群内 PPE 数	128
多段式架构参数	PPE 总数	128
	集群数	16
	每集群内 PPE 数	8

Neptune 采用多段式架构进行建模，通过配置 cluster_settings. json 文件，修改 *cluster_num*，*ppe_num* 等参数，即可实现对不同集群配置的支持，同时，Neptune 模拟将自动根据配置读取指令输入，使得不同的 PPE 集群能支持不同的处理功能，适配多段式处理结构. 一些不同架构的 cluster_settings.json 文件，如图 8 所示.

遗憾的是，由于大多数网络处理器设计者并不提供开放的架构指令集，且无法获得性能测试数据，我们难以在本工作中实现对其他网络处理器设计的适配和验证. 不过，本研究也以多段式架构中的代表产品：Netronome NFP-4000 系列处理器[18] 中的架构为例，介绍如何对 Neptune 进行修改，以支持现实中的网络处理器配置和异构处理器核.

NFP-4000 的包处理核由 PPC 核和 FPC 核组成，其中，共有 48 个 PPC 核负责网络数据的线速解析，60 个 FPC 核则被拆分成 5 个执行相同功能的集群，PPC 和 PFC 之间则相互异构. 为了对 NFP-4000 类似架构进行模拟，通过继承 PPE 类的方法创建新模块. 并通过配置 json 文件，和上文相同，通过修改集群数和集群内 PPE 数，以支持 PPC 和 FPC 的配置.

4.2 Neptune 准确性检验

在说明 Neptune 通用性的基础上，为了验证 Neptune 的准确性，我们设计了微基准测试程序和真实包处理测试程序进行验证，将生成的 2 种架构与其在真实硬件中的表现进行对照.

微基准测试程序用于验证单核性能模拟的准确性，我们使用核内指令测试集和核外访问测试集进行验证. 核内指令测试集包括算术运算、核内内存搬移等指令，用于验证 PPE 核内模拟部分的功能正确性；核外访问测试集包括读包缓冲、写包缓冲和查表指令 3 类测试，与实测性能进行比较，对单 PPE 核外访问精度进行测试.

另外，我们针对同一个真实的网络包处理流程，设计了单段式架构和多段式架构的网络处理测试集. 在单段式架构中，单核即完成全部的网络处理功能；而在多段式架构中，该测试集被拆分成了包解析、匹配执行和再封装 3 段指令集，注入不同的 PPE 集群中，与实测执行时间进行比较，以验证其准确性.

cluster_settings.json

```
{
 "cluster_num": 1,
 "clusters":[
   {
    "ppe_type": "ppe_si",
    "ppe_num": 128,
    "ins":"ins_0.asm"
   }
 ]
}
```

cluster_settings.json

```
{
 "cluster_num": 16,
 "clusters":[
   {
    "ppe_type": "ppe_mu",
    "ppe_num": 8,
    "ins":"ins_00.asm"
   },
   ...
   {
    "ppe_type": "ppe_mu",
    "ppe_num": 8,
    "ins": "ins_04.asm"
   }
 ]
}
```

cluster_settings.json

```
{
 "cluster_num": 6,
 "clusters":[
   {
    "ppe_type": "ppc",
    "ppe_num": 48,
    "ins":"ins_p.asm"
   },
   ...
   {
    "ppe_type": "fpc",
    "ppe_num":12,
    "ins": "ins_f.asm"
   }
 ]
}
```

Fig. 8　Configuration supports of different NP architectures

图 8　不同网络处理器架构的配置支持

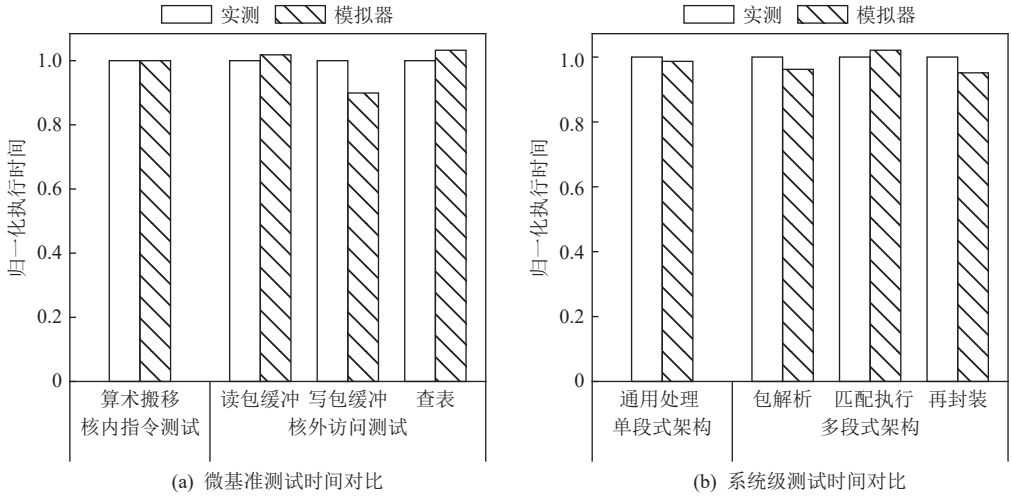
基于对核内指令和核外访问进行测试的微基准测试程序和真实应用中单段式架构和多段式架构的系统级模拟测试程序，我们对 Neptune 准确性评估结果如图 9 所示.

在微基准测试中，对于核内的算术搬移指令，由于 Neptune 框架可以对 PPE 内部结构进行精确模拟，在这一测试中，Neptune 执行时间与实际网络处理器执行时间相同. 而在外部存储进行访问时，Neptune 与访存指令则存在一定的偏差，最低为–10.1%. 这主要是由于 Neptune 还没有对总线模型进行集成，因此其只能对核外访问的周期数进行近似模拟.

而对于单段式架构和多段式架构，在对网络处理的真实应用的系统级模拟中，Neptune 模拟框架的模拟偏差同样较小. 在单段式架构的通用网络处理模拟中，模拟器执行时间为实测时间的 98.7%；而在多段式架构模拟中，模拟器在 3 个阶段与实测执行时间的偏差分别为–3.8%，+2.1%，–4.9%，整体偏差均在 5% 以内. 基于微基准测试结果，这些误差主要来自对核外访问模拟的偏差，而这些偏差在真实网络处理应用中会被进一步缓解. 网络处理器模拟器中，提供了单段式模拟准确率数据的 NePSim 中显示，结果的平均延时误差为 6%[6]，而多核模拟器 McSimA+[27] 的模拟平均延时误差为 5.7%，专用 GPU 模拟器 MGPU-Sim 中平均延时误差为 5.5%[38]. 我们认为，基于已有文献中的准确度估计，在探索软硬件优化空间的过程中，Neptune 所给出的 5% 以内的模拟误差是可接受的.

4.3　Neptune 高效性检验

为了测试 Neptune 的高效性，我们基于一台普通的个人计算机对 Neptune 进行性能测试. 测试环境的 CPU 为 6 核 Intel Core I5-12400，时钟主频为 2.5 GHz，提供 32 GB 内存. 对于模拟器，在 Linux Ubuntu 22.04 环境中以 O0 模式进行编译，设置线程数为 4，并通过



(a) 微基准测试时间对比　(b) 系统级测试时间对比

Fig. 9　Comparison of execution time between NP and Neptune

图 9　网络处理器与 Neptune 执行时间对比

函数 *omp_get_wtime*() 插桩计算其性能开销. 在该测试中, 我们让数据包以不同的形式输入 Neptune 框架中进行检验. 我们检验了 Neptune 在单数据包流经 4 个 PPE 核下取得的性能, 并逐渐增大流量到 128 个 PPE 核都被使用时的性能进行检验. 我们的性能检验结果如图 10 所示.

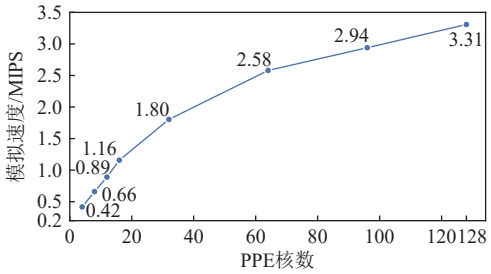


Fig. 10 Performance evaluation of Neptune simulation

图 10 Neptune 模拟性能评估

由图 10 可知, 在并行处理的 PPE 核数较少时, 随着负载的增加, Neptune 处理性能稳定增加, 在超过 16 核以后, 受不同 PPE 与外部模块交互带来的锁与同步机制, 处理速度增长逐渐减缓. 在 128 核同时处理时, Neptune 能够以总 3.31MIPS 的速度进行性能模拟. 值得一提的是, 我们的混合事件驱动与时间驱动方法保证了这一数值并不受数据包输入时间的影响, 增大数据包的输入间隔并不会对运行时间产生影响, 在 5 ms 内输入 1 000 数据包的测试中, 相比纯时钟驱动时延, 从 1.27 s 降低到 0.38 s. 在具体性能上, 若网络处理器工作频率为 800 MHz, 其模拟性能与真实性能缩放比约为 30 936 倍. 作为参照, 同为网络处理器模拟器的 PFPSim[7] 基于双核处理器对 64 核进行模拟时, 模拟性能与真实性能缩放比约为 790 292 倍. 可以看到, Neptune 相较于 PFPSim 有一个数量级的性能提升. 其他网络处理器模拟器则没有提供类似的数据. 此外, 虽然并不直接对网络处理器进行模拟, 端到端网络设备模拟器 SimBricks 在模拟时的性能缩放比约为 450 704 倍[34]; 快速多核模拟器 RCV2 中, 单 CPU 每秒所执行指令数为 413 条[39], 对应模拟速度也低于 Neptune. 可以看出, Neptune 的模拟结果与其他模拟器性能相比具有显著优势, 体现了其模拟的高效性.

5 Neptune 在架构设计中的应用分析

通过 Neptune 架构模拟和性能仿真系统, 我们能够生成单段式、多段式的架构并进行准确的性能模拟. 下面通过 3 个案例, 展示了 Neptune 如何用于架构设计和软硬件优化.

5.1 不同段式架构下包处理性能分析

在架构设计中, 面对不同段式处理器的取舍, 需要我们对不同架构的包处理性能进行评估. Neptune 通过其通用性设计, 使得架构开发者能够对不同段式的架构进行评估.

为此, 我们展示了基于表 1 中具有 128 核的单段式和多段式架构给出一个案例. 该案例中, 根据功能包处理任务最多可以拆分成包解析、匹配执行、统计限速以及再封装 4 段. 我们提出 2 种多段式架构, 分别为多段式-2 段和多段式-4 段架构, 总共设计了 3 种架构进行差异分析:

1)单段式架构. 在单段式架构中, 128 个 PPE 核同时执行包处理、匹配执行、统计限速、再封装功能.

2)多段式架构-2 段. 此时, 16 个集群被拆分成 4 和 12 共 2 段. 我们将包解析单独设置成一个任务段处理, 将其余任务交给另一段进行处理.

3)多段式架构-4 段. 此时, 16 个集群被拆分成 4, 7, 1, 4 共 4 段处理. 由于统计限速任务执行较少, 因而只设置 1 组集群.

而对于这些架构, 我们选择了 3 组网络包处理任务进行比较:

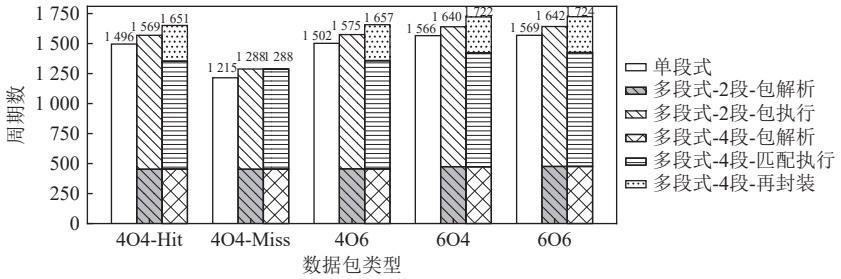
1)VXLAN 数据包. VXLAN 数据包涉及包处理任务, 包括包解析、匹配执行和再封装. 分为多种场景, 包括 IPv4-Overlay-IPv4(记作 4O4), IPv4-Overlay-IPv6(记作 4O6), IPv6-Overlay-IPv6(记作 6O6). 对于 4O4 场景, 包括匹配查表命中和不命中 2 类情况, 记作 4O4-Hit 和 4O4-Miss.

2)TCP 数据包. 常规 TCP 数据包同样涉及包解析、匹配执行和再封装处理. 一些 TCP 数据包在经过匹配后不进行再封装, 而是直接输出, 记作 TCP 和 TCP-Pass 两类. 还有一些数据包同样是 TCP 数据包, 但其处理涉及统计限速操作, 分为 Meter-Drop 和 Meter-Pass 两类.

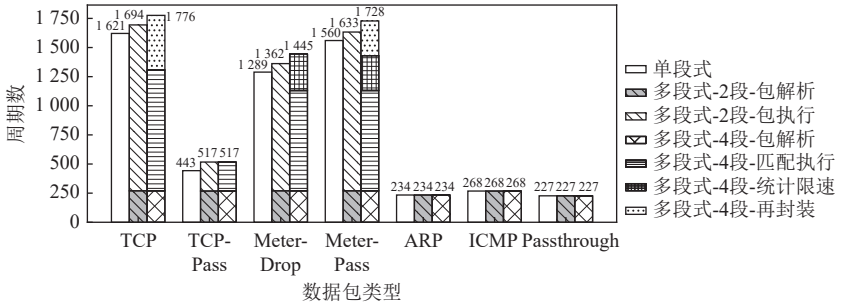
3)ARP, ICMP 数据包和直通数据包. 这 2 类数据包只进行包解析, 不进行后续操作. 一些特殊情况下, 在包解析后直接转发, 记为 Passthrough.

基于以上 3 种架构和 3 类网络数据包场景, 借助 Neptune 模拟器, 我们对数据包处理性能差异进行分析. 得到不同数据包流在单段式、多段式-2 段和多段式-4 段架构中的包处理周期数, 如图 11 所示.

图 11 结果显示, 如果仅考虑单类数据包在网络处理器中的处理性能, 单段式处理具有最好的处理



(a) 3种架构处理第1类数据包的性能评估



(b) 3种架构处理第2, 3类数据包的性能评估

Fig. 11 Performance evaluation of three NP architectures in different packets

图 11 3 种 NP 架构在不同数据包中的性能评估

性能，多段式-2 段架构次之，多段式-4 段计算架构表现性能最差. 具体而言，除了直通数据包外，在参考设计中，单段式架构较多段式-2 段架构能取得 1.045~1.167 倍的性能提升，较多段式-4 段架构能取得 1.096~1.167 倍的性能提升.

另外，我们通过 Neptune，收集了在上述数据包中多段式-2 段、多段式-4 段架构各个阶段计算占比开销，如表 2 所示. 对于多段式-2 段架构而言，在所有的 VXLAN 数据包情况中，包解析阶段与包执行阶段的执行时间比值都大于预设的 4/12. 这一问题在多段式-4 段处理架构中同样存在. 因此，当 VXLAN 数据包大量涌入时，包解析模块可能成为潜在性能瓶颈.

Table 2 Time Proportion of Packet Processing Stages for Different Multi-Phase Architecture

表 2 不同多段式架构包处理阶段时间占比

数据包类型	多段式-2 段 各段时间占比	多段式-4 段 各段时间占比
4O4-Hit	29：71	28：54：0：18
4O4-Miss	35：65	35：65：0：0
4O6	29：71	28：54：0：18
6O4	29：71	28：55：0：17
6O6	29：71	28：55：0：17
TCP	16：84	15：58：0：26
TCP-Pass	52：48	52：48：0：0
Meter-Drop	20：80	19：59：22：0
Meter-Pass	17：83	16：50：17：17
ARP	1：0	1：0：0：0
ICMP	1：0	1：0：0：0
Passthrough	1：0	1：0：0：0

利用 Neptune 性能仿真框架，我们能在设计早期发现多段式架构和单段式架构中的性能差异，并了解到各段的执行时间，从而结合目标网络场景中的网络情况，选择网络处理器不同的分段和模式进行集群数分配.

5.2 异构多段式架构性能分析

在对包处理任务进行拆分后，异构多段式架构可以针对性地对包处理架构进行优化，从而提升其处理性能. 5.1 节中结果显示，VXLAN 的包解析模块很可能成为性能瓶颈. 因此，可通过可编程流水线替换包解析段中的包处理引擎来换取更好的性能.

在包解析中，可编程流水线能根据配置，在每个流水级对数据包进行对应的包解析操作，如字段提取、跳转等. 可编程流水线周期数固定，对小层数数据包的解析缓慢，也不支持过于复杂的包解析操作，但对于层数较多的数据包，流水线相比于 PPE 解析能取得性能优势. 目前，开源的流水线包解析器 HyperParser 在 ASIC 芯片中能以 3.2~6.8 Tbps 的吞吐率进行包解析[40].

借助 Neptune 工具，我们可以对可编程流水线和包处理引擎的处理模式进行评估. 在 Neptune 中，我们实现了一个继承 PPE 类的流水线模块，通过与图 8

中相同的方式修改 Neptune 的配置，将第 1 个 PPE 集群中的配置“*ppe_type*”设置为“*ppl_parser*”，并将“*ppe_num*”设置为 1，即可得到一个用可编程流水线模块替换第 1 个 PPE 集群的异构多段式网络处理器，结构大致如图 2 所示.

为了评估基于异构流水线和同构 PPE 的包解析模块的性能，我们选取了一个包解析测试集，支持包括 VXLAN 内层数据包最多 11 层的协议解析，详细的包解析层次如图 12 所示. 我们通过 Neptune 获取了 PPE 中对上述协议进行解析的包处理引擎周期数，并编写了相应的可编程流水线配置，利用 Neptune 模拟框架，我们得到二者的比较结果如表 3 所示.

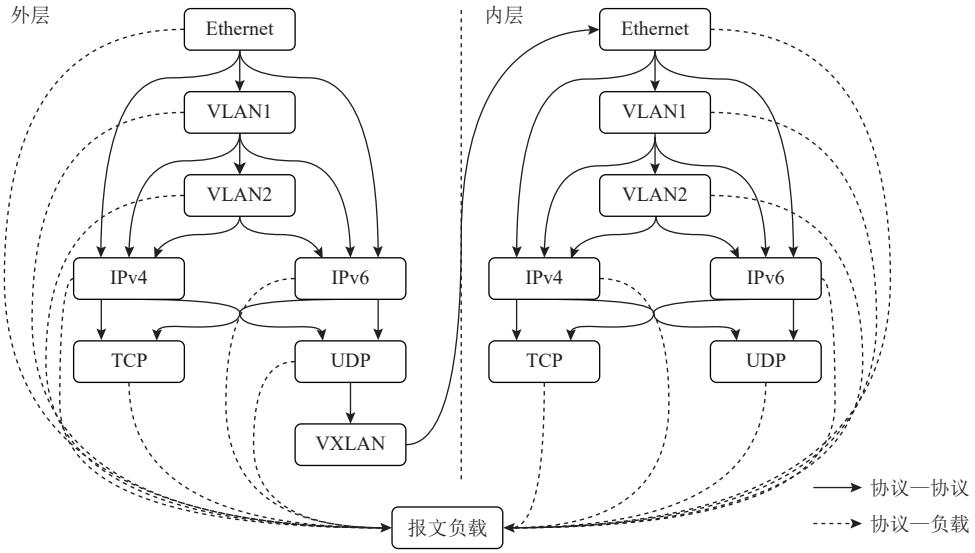


Fig. 12 Illustration of protocol hierarchy in the packet parser module test set

图 12 包解析模块测试集中的协议层次示意图

Table 3 Performance Evaluation of Packet Parser Modules in Different Protocol Layers

表 3 不同协议层数下不同包解析模块性能评估

层次数	PPE 平均周期	流水线平均周期
1	16.0	68
2	43.7	68
3	57.0	68
4	66.4	68
5	75.5	68
6	99.5	68
7	115.4	68
8	126.8	68
9	136.8	68
10	146.7	68
11	155.0	68
平均	126.8	68
VXLAN-TCP	120.0	68

从表 3 可以看到，虽然基于 PPE 的处理方法在对小于 4 层的数据包进行解析时能取得一定性能优势，但在面对较为复杂的封装如 VXLAN 网络时，PPE 处理周期几乎全面落后于较为固定的流水线.

从具体的包处理上，我们选取了 4O4 的 VXLAN 数据包进一步评估异构多段式架构带来的性能变化. 该报文头部长度为 104 B，报文如图 13 所示. 延时分析的数据显示，在包处理引擎中，为了解析 VXLAN-TCP 数据包，便需要 120 个时钟周期，且在过程中不能切换线程进行下一个包解析. 而与之相比，可编程流水线的解析周期为 68 个时钟周期，提升了 52 个时钟周期. 而根据图 11 中的数据，可编程流水线能将多段式-2 段架构和单段式架构的单包处理性能差异降低至 21 个周期. 而从吞吐量的计算上，可编程流水线可以在 1 个时钟周期内处理 32 B 的包头内容. 通过以下公式，可计算可编程流水线相比于包解析器的吞吐率比值为：

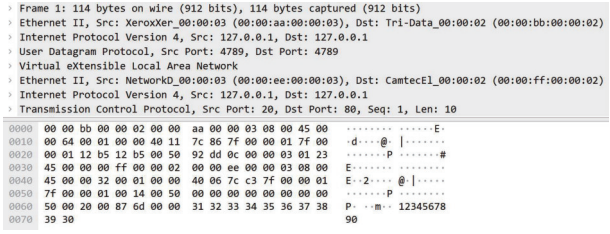


Fig. 13 Hierarchy of VXLAN-TCP data packet

图 13 VXLAN-TCP 数据包层次

$$\frac{Process_bytes_{\text{Pipeline}}/Cycles_{\text{Pipeline}}}{Process_bytes_{\text{PPE}}/Cycles_{\text{PPE}}}=36.92.$$

基于此，可编程流水线吞吐约为单个包处理引擎的 36.92 倍. 要达到与流水线相近的处理性能，需要至少 37 个 PPE，这一结果也与 Zolfaghari 等人[23]基于硬件的评估结果相近.

5.3 多线程包处理服务质量分析

增加网络处理器中每个 PPE 的线程数，能使得 PPE 与外部模块交互带来的延时开销被覆盖，从而提升网络处理器处理的平均延时与吞吐. 然而，更高的线程数也意味着需要更大的包缓冲区，过多的线程数既不会带来吞吐的进一步提升，又会导致部分数据包的处理延时激增，影响服务质量. 在多段式中，分段能使得服务质量得到更好的保障；而在单段式架构中，这一问题则更加明显，需要软硬件设计人员协同合作，探索最适合的线程数并进行相应的优化.

借助 Neptune，我们同样可以比较 PPE 在不同线程数下的包处理性能. 我们编写了一个单段式参考的多线程事例，其中涉及 1 次包读取、2 次包写入操作. 在 PPE 执行核外操作时，进行线程切换以覆盖这一延时开销. 当输入数据包相同时，我们得到的 PPE 平均处理延时和最大处理延时如图 14 所示.

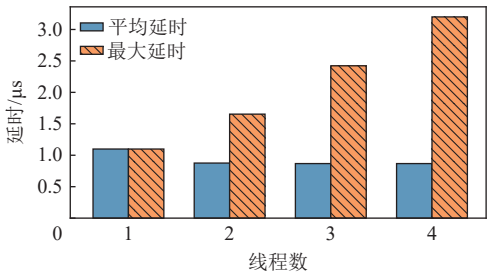


Fig. 14 Latency evaluation of packet processing under different thread counts

图 14 不同线程数下包处理延时评估

由图 14 可以得知，在该 PPE 和微码的环境下，当线程数为 3 时，PPE 的平均处理延时达到最小，此后，增加额外的线程数并不能带来处理延时的提升. 另一方面，随着线程数的增加，包处理的最大延时也在不断增加. 因此，对于大数据包而言，增大单段式处理的线程数很可能导致大数据包的处理时间违反服务质量需求，在该案例中 2 个线程是一个较为合理的数值.

综上，基于 Neptune 模拟框架，我们可以对网络处理器架构进行全面探索，既能对单段式、多段式架构的包处理性能进行分析，又能对异构多段式的硬件优化、多线程设计中的软件优化进行评估.

6 讨 论

虽然 Neptune 框架能够对网络处理器的软硬件设计提供指导和参考作用，但也需指出，当前 Neptune 同样也存在其他可进一步改进之处.

由于 Neptune 需要为硬件验证提供一定程度上的前期辅助，因而在上述用例中，Neptune 对 PPE 的模拟都是细化到流水级程度. 这虽然使得 PPE 能完全精准地反映算术周期，但也导致网络处理器模拟的性能开销增加. 不过，Neptune 框架同样也提供了精简模拟的可能，通过对 PPE 模拟结构的简化，我们可以用轻微的准确度损失换取更好的模拟性能.

目前，Neptune 在对外部事件的模拟中，仍依赖于硬件仿真中得到的粗粒度数据. 这可能导致 Neptune 在对外部模块的模拟存在一定程度的偏差. 我们的下一步工作将包括整合总线与存储的相关模拟工作，以更好地对外部访问进行模拟.

7 总 结

由于以太网带宽的高速增加和网络包处理任务与硬件更新速度的不匹配，网络数据面的高性能和可编程成为一大重要需求. 网络处理器作为一种专用处理器，为网络包处理提供了有力的平台. 然而，在已有的网络处理器架构中，单段式架构和多段式架构同时存在，网络处理器设计人员希望在设计早期就对此进行架构探索. 对此，本文提出了一种通用网络处理器的结构模拟和性能仿真框架 Neptune. 通过分析，Neptune 采用多段式架构支持 2 种网络处理器架构，并采用事件链表和核间队列对数据通路与多段式架构的核间调度进行支持以保障通用性. 还采用同步图计算和结合事件驱动思想，保障了模拟准确性和高效性. Neptune 通过配置支持不同的网络处理器架构模拟，在真实应用模拟中，Neptune 能达到 95% 以上的性能模拟准确率. 在性能模拟中，取得 3.3 MIOPs 的性能，相比于过去的 PFPSim 网络模拟器，存在一个数量级的性能提升.

最后，我们展示了 Neptune 在架构设计中的应用. 其能对参考设计中的单段式架构和多段式架构进行分析，也可以评估异构化设计的多段式架构对性能带来的改变，还能对网络处理器核内线程数设置提供参考. 进一步工作将包括整合总线、存储相关模拟工作、整合功耗模拟等.

作者贡献声明:林涵越提出了系统设计和实验方案,并撰写了论文内容;吴婧雅对论文提供了关键的修改意见;卢文岩对论文选题提供了关键指导意见;钟浪辉对论文实验部分提供了测试平台,并对该部分修改给出了关键意见;鄢贵海对论文选题提供指导意见,审阅与修订了论文.

参 考 文 献

[1] Gadre G, Badhe S, Kulkarni K. Network processor—A simplified approach for transport layer offloading on NIC[C]//Proc of the 2016 Int Conf on Advances in Computing, Communications and Informatics (ICACCI). Piscataway, NJ: IEEE, 2016: 2542−2548

[2] Yang Mingran, Baban A, Kugel V, et al. Using trio: Juniper networks' programmable chipset-for emerging in-network applications[C]//Proc of the ACM SIGCOMM 2022 Conf. New York: ACM, 2022: 633−648

[3] Krude J, Rüth J, Schemmel D, et al. Determination of throughput guarantees for processor-based smartnics[C]//Proc of the 17th Int Conf on Emerging Networking Experiments and Technologies. New York: ACM, 2021: 267−281

[4] Zhao Yuyu, Cheng Guang, Liu Xuhui, et al. Survey and applications of next generation network processor[J]. Journal of Software, 2021, 32(2): 445−474 (in Chinese)
(赵玉宇, 程光, 刘旭辉, 等. 下一代网络处理器及应用综述[J]. 软件学报, 2021, 32(2): 445−474)

[5] Yan Guihai, Lu Wenyan, Li Xiaowei, et al. Comparative study of the domain-specific processors[J]. SCIENTIA SINICA Informationis, 2022, 52(2): 358−375 (in Chinese)
(鄢贵海, 卢文岩, 李晓维, 等. 专用处理器比较分析[J]. 中国科学: 信息科学, 2022, 52(2): 358−375)

[6] Luo Yan, Yang Jun, Bhuyan L N, et al. NePSim: A network processor simulator with a power evaluation framework[J]. IEEE Micro, 2004, 24(5): 34−44

[7] Abdi S, Aftab U, Bailey G, et al. PFPSim: A programmable forwarding plane simulator[C]//Proc of the 2016 Symp on Architectures for Networking and Communications Systems. New York: ACM, 2016: 55−60

[8] Bosshart P, Gibb G, Kim H S, et al. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN[J]. ACM SIGCOMM Computer Communication Review, 2013, 43(4): 99−110

[9] Moon Y G, Lee S E, Jamshed M A, et al. AccelTCP: Accelerating network applications with stateful TCP offloading[C]//Proc of the 17th USENIX Symp on Networked Systems Design and Implementation (NSDI'20). Berkeley, CA: USENIX Association, 2020: 77−92

[10] Choi S, Shahbaz M, Prabhakar B, et al. λ-nic: Interactive serverless compute on programmable smartnics[C]//Proc of the 40th Int Conf on Distributed Computing Systems (ICDCS). Piscataway, NJ: IEEE, 2020: 67−77

[11] Xi Shaoke, Li Fuliang, Wang Xingwei. FlowValve: Packet scheduling offloaded on NP-based SmartNICs[C]//Proc of the 42nd Int Conf on Distributed Computing Systems (ICDCS). Piscataway, NJ: IEEE, 2022: 347−358

[12] Hypolite J, Sonchack J, Hershkop S, et al. DeepMatch: Practical deep packet inspection in the data plane using network processors[C]//Proc of the 16th Int Conf on Emerging Networking Experiments and Technologies. New York: ACM, 2020: 336−350

[13] Cisco. Cisco Silicon One P100 processor data sheet [EB/OL]. (2021-10-25)[2024-01-18]. https://www.cisco.com/c/en/us/solutions/collateral/silicon-one/silicon-one-p100-processor-ds.html

[14] Vlachos K, Orphanoudakis T, Papaeftathiou Y, et al. Design and performance evaluation of a programmable packet processing engine (PPE) suitable for high-speed network processors units[J]. Microprocessors and Microsystems, 2007, 31(3): 188−199

[15] Liu Siyuan, Ren Minhua, Gu Hangping. Design of network processor micro-engine based on hardware multi-threading mechanism[J]. Microcomputer Application, 2022, 38(2): 106−108 (in Chinese)
(刘思远, 任敏华, 谷航平. 基于硬件多线程机制的网络处理器微引擎设计[J]. 微型电脑应用, 2022, 38(2): 106−108)

[16] Chole S, Fingerhut A, Ma Sha, et al. dRMT: Disaggregated programmable switching[C]//Proc of the 2017 Conf of the ACM Special Interest Group on Data Communication. New York: ACM, 2017: 1−14

[17] Sundar N, Burres B, Li Yadong, et al. 9.4 An in-depth look at the Intel IPU E2000[C]//Proc of the 2023 IEEE Int Solid-State Circuits Conf (ISSCC). Piscataway, NJ: IEEE, 2023: 162−164

[18] Netronome. NFP−4000 theory of operation[EB/OL]. 2018[2024-01-18]. https://d3ncevyc0dfnh8.cloudfront.net/media/documents/WP_NFP4000_TOO.pdf

[19] Yazdinejad A, Parizi R M, Bohlooli A, et al. A high-performance framework for a network programmable packet processor using P4 and FPGA[J]. Journal of Network and Computer Applications, 2020, 156: 102564

[20] Li Tao, Yang Hui, Li Junnan, et al. ChipletNP: Chiplet-based agile customizable network processor architecture[J]. Journal of Computer Research and Development, 2024, 61(12): 2952−2968
(李韬, 杨惠, 厉俊男, 等. ChipletNP: 基于芯粒的敏捷可定制网络处理器架构[J]. 计算机研究与发展, 2024, 61(12): 2952−2968)

[21] Ahmadi M, Wong S. A performance model for network processor architectures in packet processing system[C]//Proc of the 19th IASTED Int Conf on Parallel and Distributed Computing and Systems. Calgary, AB, Canada: ACTA Press, 2007: 176−181

[22] Keslassy I, Kogan K, Scalosub G, et al. Providing performance guarantees in multipass network processors[J]. IEEE/ACM Transactions on Networking, 2012, 20(6): 1895−1909

[23] Zolfaghari H, Mustafa H, Nurmi J. Run-to-completion versus pipelined: The case of 100 Gbps packet parsing[C/OL]//Proc of the 22nd Int Conf on High Performance Switching and Routing (HPSR). Piscataway, NJ: IEEE, 2021[2024-01-18]. https://ieeexplore.ieee.org/abstract/document/9481797

[24] Wehrie K, Gunes M, Gross J. Modeling and Tools for Network Simulation[M]. Berlin: Springer, 2010

[25] Fan Chengze, Bi Jun, Zhou Yu, et al. NS4: A P4-driven network simulator[C]//Proc of the 2017 SIGCOMM Posters and Demos. New York: ACM, 2017: 105−107

[26] Gao Kaihui, Chen Li, Li Dan, et al. Dons: Fast and affordable discrete event network simulation with automatic parallelization[C]//Proc of the ACM SIGCOMM 2023 Conf. New York: ACM, 2023: 167−181

[27] Ahn J H, Li Sheng, Seongil O, et al. McSimA+: A manycore simulator with application-level+ simulation and detailed microarchitecture modeling[C]//Proc of the 2013 IEEE Int Symp on Performance Analysis of Systems and Software (ISPASS). Piscataway, NJ: IEEE, 2013: 74−85

[28] Ren Pengju, Lis M, Cho M H, et al. HORNET: A cycle-level multicore simulator[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2012, 31(6): 890−903

[29] Qureshi Y M, Simon W A, Zapater M, et al. Gem5-X: A gem5-based system level simulation framework to optimize many-core platforms[C/OL]//Proc of the 2019 Spring Simulation Conf (SpringSim). Piscataway, NJ: IEEE, 2019[2024-01-18]. https://ieeexplore.ieee.org/abstract/document/8732862

[30] Arashloo M T, Lavrov A, Ghobadi M, et al. Enabling programmable transport protocols in high-speed NICs[C]//Proc of the 17th USENIX Symp on Networked Systems Design and Implementation (NSDI'20). Berkeley, CA: USENIX Association, 2020: 93−109

[31] Wagner J, Leupers R. A fast simulator and debugger for a network processor[C/OL]//Proc of Embedded Intelligence Conf. 2002[2024-03-21]. https://www.researchgate.net/publication/228724737_A_fast_simulator_and_debugger_for_a_network_processor

[32] Koohi M, Bayadi H, Khaless M N. A simulation environment for network processor based on simultaneous multi thread architecture[J]. Indian Journal of Science and Technology, 2012, 5(10): 1−6

[33] Bosshart P, Daly D, Gibb G, et al. P4: Programming protocol-independent packet processors[J]. ACM SIGCOMM Computer Communication Review, 2014, 44(3): 87−95

[34] Li Hejing, Li Jialin, Kaufmann A. SimBricks: End-to-end network system evaluation with modular simulation[C]//Proc of the ACM SIGCOMM 2022 Conf. New York: ACM, 2022: 380−396

[35] Netronome. Programmer studio 6.0[EB/OL]. 2016[2024-03-18]. https://d1agld16eywpip.cloudfront.net/media/documents/PB_Programmer_Studio_6.0_rURUo4Y.pdf

[36] Sokolowski J A, Banks C M. Modeling and Simulation Fundamentals: Theoretical Underpinnings and Practical Domains[M]. Hoboken, NJ: John Wiley & Sons, 2010

[37] Shah N, Kurt K. Network processors: Origin of species[C]//Proc of the 17th Int Symp on Computer and Information Science (ISCIS XVII). Boca Raton, FL: CRC, 2002: 41−45

[38] Sun Yifan, Baruah T, Mojumder S A, et al. MGPUSim: Enabling multi-GPU performance modeling and optimization[C]//Proc of the 46th Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2019: 197−209

[39] Guo Xuan, Mullins R. Accelerate cycle-level full-system simulation of multi-core RISC-V systems with binary translation[J]. arXiv preprint, arXiv: 2005.11357, 2020

[40] Liu Huan, Qiu Zhiliang, Pan Weitao, et al. HyperParser: A high-performance parser architecture for next generation programmable switch and SmartNIC[C]//Proc of the 5th Asia-Pacific Workshop on Networking (APNet 2021). New York: ACM, 2021: 50−56



Lin Hanyue, born in 1999. PhD candidate. Member of CCF. His main research interests include domain-specific computer architecture and network computing systems.

林涵越, 1999 年生. 博士研究生. CCF 会员. 主要研究方向为专用计算机体系结构与网络计算系统.



Wu Jingya, born in 1994. PhD. Member of CCF. Her main research interests include domain-specific computer architecture and heterogeneous computing system optimization.

吴婧雅, 1994 年生. 博士. CCF 会员. 主要研究方向为专用计算机体系结构和异构计算系统优化.



Lu Wenyan, born in 1990. PhD, associate professor, master supervisor. Member of CCF. His main research interests include deep learning accelerator, database accelerator, domain-specific computer architecture, and heterogeneous computing system optimization.

卢文岩, 1990 年生. 博士, 副研究员, 硕士生导师. CCF 会员. 主要研究方向为深度学习加速器、数据库加速器、专用计算机体系结构、异构计算系统优化.



Zhong Langhui, born in 1974. PhD, senior engineer. Member of CCF. His main research interests include low latency technology and securities quotation processing.

钟浪辉, 1974 年生. 博士, 高级工程师. CCF 会员. 主要研究方向为低时延技术、证券报价处理.



Yan Guihai, born in 1982. PhD, professor, PhD supervisor. Member of CCF. His main research interests include computer architecture, domain-specific accelerator design, and intelligent chip architecture.

鄢贵海, 1982 年生. 博士, 研究员, 博士生导师. CCF 会员. 主要研究方向为计算机体系结构、专用加速器设计、智能芯片体系结构.