基于 SDN 的 OpenFlow 交换机数据包流水线处理机制

吴 琪[1] 王兴伟[1] 黄 敏[2]

(东北大学软件学院 沈阳 110819)[1] (东北大学信息科学与工程学院 沈阳 110819)[2]

摘 要 目前,软件定义网络(Software Defined Networking,SDN)已成为网络研究与开发的重点,但相关的研究与开发工作还仅仅局限于园区网络和数据中心网络等。由于 SDN 控制层与数据层处理效率的限制,SDN 面向互联网这样超大规模网络的研究还基本处于空白阶段。为了提升 SDN 的性能以使其适应大规模网络的特点,挖掘 SDN 数据层中并行加速处理的可能性,提出了将流水线技术应用到 SDN 数据层中交换机对数据包的转发过程。另外,结合 SDN 南向接口 OpenFlow 协议提供的交换机工作规范,设计了适用于 OpenFlow 交换机数据包转发的三级流水线处理机制。仿真实验说明,将流水线应用到 SDN 中能有效加快 OpenFlow 交换机的数据包转发速度。

关键词 软件定义网络,OpenFlow 协议,流水线技术,数据包转发

中图法分类号 TP393 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2018.10.055

OpenFlow Switch Packets Pipeline Processing Mechanism Based on SDN

WU Qi[1] WANG Xing-wei[1] HUANG Min[2]

(College of Software,Northeastern University,Shenyang 110819,China)[1]

(College of Information Science and Engineering,Northeastern University,Shenyang 110819,China)[2]

Abstract Currently,SDN (Software Defined Networking) has become the focus of the research and development in network field,but its related research and development are limited in the campus network and data center network. Due to the limitation of the processing efficiency of control layer and data layer,the research on ultra large scale network such like Internet is basically in the blank stage. In order to improve the performance of SDN and make it adaptable to the large-scale network,this paper explored the possibility of parallel acceleration processing in SDN data layer,in which pipeline technology is applied to the packet forwarding process of OpenFlow switch in SDN data layer. Combined with SDN work specification provided by the south interface protocols OpenFlow,a 3 level pipeline processing mechanism was designed to adaptable for OpenFlow switch packets transmission. The design and simulation of this system prove that using pipeline into SDN field can speed up the packet forwarding speed of OpenFlow switch effectively.

Keywords Software defined networking,OpenFlow protocal,Pipeline technology,Packet transmission

1 引言

传统类型网络的控制器不能对全局进行掌握,只能在自己的能力范围内转发。然而,当前用户对网络的需求越发趋于多元化和定制化,这使得当前网络面临着巨大挑战。在这样的背景下,未来互联网及其相关技术成为新的研究热点。斯坦福大学 Clean Slate 项目中提出的软件定义网络[1](Software Defined Networking,SDN)以其简洁的网络架构和极强的兼容性受到学术界的广泛关注,并得到了网络设备制造商的支持,成为网络领域研究与开发的重点[2]。SDN 强调控制平面和数据平面分离、程序化实现定制需求、快速实施和部署网络策略等[3]。

在 SDN 的研究与发展历程中,越来越多的理论和实践问题不断涌现。SDN 控制与数据解耦的特点可使其与其他网络研究热点的特点整合部署,从而适应不同的网络环境。例如,SDN 与网络功能虚拟化(Network Function Virtualization)[4-5]和信息中心网络(Information Center Networking)[6]的结合一直是该研究方向的重点内容之一。而对于软件定义网络自身的特点而言,其集中化管理模式在提供多样化服务定制等优点的同时也容易导致性能瓶颈的出现,特别是在大规模以及超大规模网络中它很可能成为制约 SDN 成功应用的因素,因此需要研究与开发性能优化机制,以提高 SDN 控制平面和数据平面的可扩展性与工作效率。

关于 SDN 性能优化机制的研究与开发已经取得了一些可喜的成果,文献[7-9]提出从控制层面对 SDN 性能进行优化的方案。数据平面的性能优化同样重要。与传统网络相比,SDN 的转发规则更加复杂,分组到达时,需要支持更多的匹配域,提供更灵活的处理操作。文献[10-13]侧重于优化数

到稿日期:2017-12-14 返修日期:2018-02-26 本文受国家自然科学基金(61572123),国家杰出青年科学基金资助项目(71325002)资助。

吴 琪(1994-),女,硕士生,主要研究方向为软件定义网络、并行与分布式计算,E-mail:wuqi_kiki@foxmail.com;王兴伟(1968-),男,教授,博士生导师,主要研究方向为未来互联网、软件定义网络、网络安全等,E-mail:wangxw@mail.neu.edu.cn(通信作者);黄 敏(1968-),女,教授,博士生导师,主要研究方向为企业物流与供应链管理、现代集成制造系统、智能优化等,E-mail:huangm@mail.neu.edu.cn。

据平面以提升网络性能。上文中提到的性能优化机制多局限于仿真实现、原型实现或者小规模网络部署[13]，还没有深入研究适合大规模特别是超大规模网络需要的性能优化机制[14]，这是 SDN 领域亟待突破的方面。

OpenFlow 协议[15]是一种认知度较高的 SDN 南向协议(控制器与交换机间的通信协议)。它不仅可以提供多粒度的网络控制，简化网络管理，加快网络部署速度，还可以为用户提供编程环境，从而实现差异化服务。流水线技术是指在程序执行时多条指令重叠进行操作的一种准并行处理实现技术[16]。流水线技术是 Intel 首次在 486 芯片中开始使用的。它的工作方式就像工业生产上的装配流水线，多个部件并行处理，共同协作完成同一项任务。

本文的贡献是基于 SDN 网络环境，结合 OpenFlow 协议规定的交换机工作规范，采用流水线技术，改进 OpenFlow 交换机对多个数据包的处理流程，对数据平面处理数据包的方式进行优化，进一步提高 SDN 网络的效率，从而提高其处理互联网中海量数据请求的能力。

2 数据包流水线处理机制

SDN 主张将控制层与转发层分离。控制器通过提供编程功能来实现策略的动态和个性化部署，并负责下发转发路径给交换机，交换机只负责按照控制器下发流表转发数据包。OpenFlow 协议作为控制器和交换机之间的通信协议，对 SDN 的部署和研究有着重要意义，它可以让研究者在异构环境下用统一的方法进行实验，同时网络供应商也无需暴露其交换机的内部工作方式。OpenFlow 协议 1.10 版本[17]中规定了交换机对数据包的处理过程，OpenFlow 交换机在接收到一个数据包时，首先对数据包的头字段进行解析，存在诸如“如果是 IPv4 数据包，则以太网帧的以太网类型为 0x0800”等依赖关系。因此在本版本中要按照数据包的种类去执行头字段的解析。头字段的解析结果可以作用于流表的匹配域，数据包在流水线中从第一个流表开始依次匹配流表，若找到匹配项，则根据指令集中的指令对数据包进行操作。若指令集中有跳转表指令，则根据指令跳转到该表后继续流水线处理；若无跳转表指令，则结束流水线处理并执行行动集。若数据包在流水线中依次匹配所有流表后都没有匹配项，则判断是否有 Table-miss 表项存在。若存在则按照 Table-miss 表项中的指令集处理数据包(本版本中 Table-miss 表项默认丢弃数据包)，若无 Table-miss 表项则将数据包存到寄存器中并与控制器通信，按照控制器发回的信息处理数据包。

采用流水线技术将上述处理过程拆分成相连的流水段，不同功能的流水段并行处理多个数据包。在相邻流水段间加入中间寄存器，中间寄存器可以在相邻的两个流水段之间传送数据，以保证提供后续要用到的数据，同时把各段的处理工作互相隔离。本系统中主要包含如下组件。

ACPT：接收数据包流水段。负责并发解析数据包头字段。

MTCH：匹配多级流表的流水段。调用流表资源，并发匹配数据包和多级流表。

PROC：执行指令和行动的流水段。负责并发执行对修改数据包或流水线处理、跳转表、与数据包通信和转发或丢弃数据包等操作。

REG_{am}：存储从 ACPT 流水段出来但还未进入到 MTCH 流水段的数据包的头字段解析结果信息。

REG_{mpl}：存储从 MTCH 流水段出来但还未进入到 PROC 流水段的数据包匹配结果信息。

REG_{pm}：存储从 PROC 流水段中出来将要再次进入到 MTCH 流水段的数据包信息。

REG_{mp2}：存储从 MTCH 流水段中出来将要再次进入到 PROC 流水段的数据包匹配信息，或控制器下发的对未匹配到流表项数据包的处理行为信息。

数据包流水线处理机制的架构如图 1 所示。

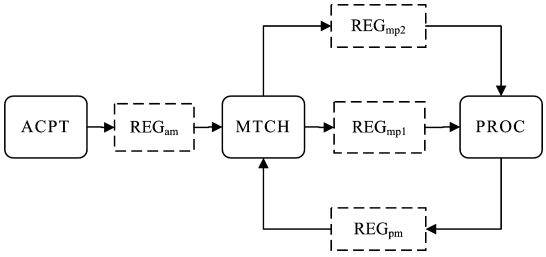

图 1 数据包流水线处理系统架构图

Fig. 1 Packet pipeline processing system architecture

2.1 流水段划分

本文将一个完整的过程拆分成一条三级流水线处理过程，3 个流水段协同工作，并发完成任务。如图 1 所示，系统中共包含 ACPT，MTCH 和 PROC 3 个流水段。

ACPT 流水段负责从端口接收数据包，解析数据包的头字段，并将解析出的数据和数据包交由下一流水段处理。MTCH 流水段负责处理从 ACPT 流水段或 PROC 流水段传来的数据包信息，利用其中的头字段解析信息与流表项中的匹配域在多级流表中依次进行匹配。在原有匹配机制中，数据包在多级流表中依次匹配直至匹配失败或指令集中没有跳转表指令才结构匹配阶段。与原有匹配机制不同，在本系统的 MTCH 流水段中，数据包一旦匹配成功则立即将匹配结果发往下一流水段，即使有跳转表指令，也在完成下一流水段指令后再返回本流水段进行匹配。PROC 流水段负责处理 MTCH 流水段发来的匹配结果或者控制器下发的消息，按照信息中的指令集或动作集(指令集中无跳转表指令时，执行动作集)对多级流表或数据包进行相应的修改操作，包括修改动作集、跳转到下一流表、转发数据包、丢弃数据包、使用指定组操作数据包、修改数据包、对流表进行操作和与控制器通信等操作。本流水段处理的控制器下发信息只包括 Flow_Mod 消息和 Packet_Out 消息，其他消息不涉及数据包处理过程，本系统不予讨论。

从上述分析中可以看出，多表流水线的 MTCH 流水段和 PROC 流水段是可反复、有数据交互的。这是因为每个数据包通过多表流水线的方式不同，同一时间在不同流水段上并行处理数据包的数量也有可能不相等。因此，不能单纯地将多个数据包平均分组后依次通过流水线。

由此，本文规定了同一个流水段并行处理多个数据包时的处理原则。

先结束先通过原则：在同一流水段并发处理的多个数据包先被处理完毕的可优先进入下一流水段中。分别最多有 n_1，n_2，n_3 个数据包在 Q_{acpt}，Q_{mtch}，Q_{proc} 中被处理。其中，n_1，

n_2,n_3 是每个流水段处理能力的最大值;Q_{acpt},Q_{mtch},Q_{proc} 分别指代接收数据包的流水段、匹配多级流表的流水段以及执行指令集和动作集的流水段。

在原交换机处理数据包机制中,数据包是串行处理的,因此在 MTCH 流水段和 PROC 流水段中,单个数据包应单独占用所有多级流表资源;而在本系统中,我们不仅将匹配和执行功能划分开,更是将每个数据包的处理过程都区分开来,因此对于流表资源的调用就产生了竞争。这里我们定义了流表资源的冲突,并给出了解决冲突的处理原则。

冲突:在并行处理多个数据包时,Q_{mtch} 和 Q_{proc} 两个流水段会出现多个数据包匹配同一个流表或对同一流表进行操作的情况出现。这种情况被称作发生冲突。

冲突的处理原则:若冲突发生在 Q_{mtch},则同时进行匹配。若冲突发生在 Q_{proc},则后到的数据包在队列中则等待。同一流表不能同时在 Q_{mtch} 和 Q_{proc} 中被调用,先到的数据包先处理,后到的数据包则等待。对于同在 Q_{mtch} 和 Q_{proc} 中等待的数据包,先进入 Q_{acpt} 的先处理。

如此设计的理由是,Q_{mtch} 只调用流表数据,不对流表表项内容进行修改,因此 Q_{mtch} 调用流表可以不加锁,多个数据包可同时匹配同一流表。而 Q_{proc} 涉及到对流表内容进行修改等操作,若不单独调用流表资源则会产生一系列的并发问题,如丢失更新、未确定相关性(脏读)和不一致性分期(非重复读)等。因此,我们允许 Q_{proc} 处理数据包时单独调用流表资源。同时,考虑到数据包自身的优先级问题,当流表资源被释放时,同时在 Q_{mtch} 和 Q_{proc} 中等待该资源的数据包,先通过 Q_{acpt} 的数据包先获取资源。

2.2　寄存器划分

图 1 所示的 4 个寄存器分别存储 4 种不同状态的数据包信息。

REG_{am} 和 REG_{pm} 分别存储首次和再次进入 Q_{mtch} 的数据包。REG_{mp1} 和 REG_{mp2} 分别存储首次和再次进入 Q_{proc} 的数据包。所存储的数据包流向相同流水段的多个寄存器被称为同级别寄存器。在流水线的 Q_{mtch} 中,同阶段会发生调用线程资源冲突。处于同一阶段的寄存器的优先级不同,REG_{pm} 的优先级高于 REG_{am},REG_{mp2} 的优先级高于 REG_{mp1}。

由于匹配资源的处理能力有限,即同一时刻最多只能有一定数量的数据包在 Q_{mtch} 中被处理,因此当同一级别的寄存器的两个寄存器都有数据包等待分配线程资源时,优先级高的寄存器中的数据包优先被处理,只有高优先级的寄存器中没有数据包等待时,低优先级寄存器中的数据包才能进入到下一流水段。

这样设计的优点是,开通旁路,增加寄存器数量,有利于减轻流水线压力,减轻资源积压现象,加快流水线的周转速度。

3　性能评价

为了对 OpenFlow 交换机数据包流水线处理机制进行性能测试,我们在 Mininet 仿真平台上搭建仿真环境并远程连接 FloodLight 控制器实现上述数据包流水线处理机制。在 Mininet 仿真平台上搭建拓扑环境。实验中,在 FloodLight 控制器的 PC 机上启动 Wireshark,抓取当前网络环境下的数据包,对当前网络环境进行监控,从而查看采用不同流水线机制的交换机对数据包的处理效率。本文的性能评价指标如下。

(1)加速比

加速比 $Sp_{(q)}$ 的计算公式为:

$$Sp_{(q)}=\frac{T_s}{Tp_{(q)}} \tag{1}$$

其中,T_s 是串行算法在一个处理器上的执行时间,$Tp_{(q)}$ 是并行算法在 q 个处理器上的执行时间。

加速比分为最大加速比和实际加速比。系统的实际加速比主要与流水段的处理时间、缓存寄存器的延迟时间有关,流水段的处理时间越长,缓存寄存器的延迟时间越大,那么这条流水线的实际加速比就越小。

当流水线各段处理时间不相同时,一条 k 段流水线完成 n 个连续任务的实际加速比公式 Sp 可以表示为:

$$Sp=\frac{n}{\sum_{i=1}^{k}\Delta t_i+(n-1)*\max(\Delta t_1,\Delta t_2,\cdots,\Delta t_k)} \tag{2}$$

其中,n 是处理的任务包数,k 是流水段数,Δt_i 表示第 i 段的处理时间。式(2)中 n 趋近于正无穷时,可推导得出最大加速比 Sp_{max}:

$$Sp_{max}=\frac{1}{\max(\Delta t_1,\Delta t_2,\cdots,\Delta t_k)} \tag{3}$$

(2)吞吐率

吞吐率是计算机中的流水线在单位时间内可输出的数据量。吞吐率 TP_K 的计算公式可以表示为:

$$TP_K=\frac{n}{T_K} \tag{4}$$

其中,n 是系统当前处理完的任务数量,T_K 是处理完 n 个任务所用的时间。

当流水线各段处理时间不相同时,一条 k 段流水线完成 n 个连续任务的实际吞吐率 TP 为:

$$TP=\frac{n}{\sum_{i=1}^{k}\Delta t_i+(n-1)*\max(\Delta t_1,\Delta t_2,\cdots,\Delta t_i)} \tag{5}$$

其中,n 是处理的任务包数,k 是流水段数,Δt_i 表示第 i 段的处理时间。式(5)中 n 趋近于正无穷时,可推导得出最大吞吐率 TP_{max}:

$$TP_{max}=\frac{1}{\max(\Delta t_1,\Delta t_2,\cdots,\Delta t_k)} \tag{6}$$

(3)效率

效率即各流水段的使用效率。流水线中设备的实际使用时间与整个运行时间的比值就是流水线设备的利用率。

由于流水线有通过时间、排空时间和额外时间开销,因此在连续完成 n 个任务的时间内,各段并不是满负荷工作的。用流水段线程的工作时间除以整条流水线的工作时间就是这条流水线的效率。

通过时间:第一个任务从进入流水线到流出结果所需的时间。

排空时间:最后一个任务从进入流水线到流出结果所需的时间。

额外时间开销:流水寄存器延迟和时钟偏移的时间开销。

缓存寄存器的时间开销包括建立时间和传输延迟。建立时间是在接触写操作的时钟信号到达之前,寄存器输入必须

保持稳定的时间。传输延迟始终是信号到达到寄存器输出可用的时间。

在 Mininet 仿真平台上搭建的拓扑环境中，对上文提到的数据包流水线处理机制连续处理 1000 个数据包的吞吐率、加速比和效率 3 项性能指标进行了测量。测量结果分别如图 2－图 4 所示。

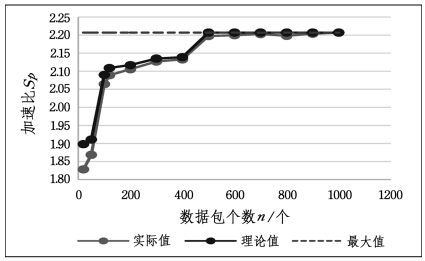

图 2 3 个加速比的趋势变化图

Fig. 2 Trend map of three acceleration ratios

图 2 所示为系统模拟连续处理 1000 个数据包时，加速比的实际值、理论值和最大值的变化示意图。可以看出，本系统的流水线加速比的最大值为 2.207。当系统处理的数据包较少，即数据包数目在 500 以下时，加速比较少，这是由于处理的任务数较少，流水线内线程有闲置情况，因此随着数据包数目的增大，加速比的实际值和理论值都逐渐提升。同时，在该段区域内，加速比的实际值与理论值之间的差距开始时较大，随着数据包数量的增加，差距逐渐缩小。这是因为在流水线启动阶段，存在系统通过时间、排空时间和额外时间等时间开销（如流表初始化和读取寄存器等），当数据包数量较小（少于 200 个）时，这些时间所占的比例较大，因此实际值与理论值的差距较大。但随着数据包数量的上升，流水线系统的有效处理时间所占比例加大，实际值与理论值的差距也越来越小。

当数据包数量足够大（多于 500 个）时，理论值达到最大值 2.207，实际加速比也趋于稳定，且在略小于最大值的位置浮动。流水线满负荷运作时，实际加速比和理论值会达到最大值并趋于稳定。实际值与理论值的差距是由通过时间、排空时间和额外时间开销导致的。

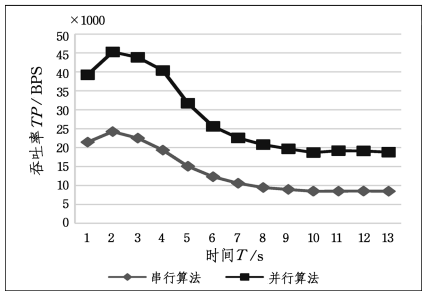

图 3 串行算法与并行算法吞吐率的对比

Fig. 3 Comparison of throughputs of serial algorithm and parallel algorithm

图 3 给出流水线并行算法和串行算法吞吐率的对比结果。从图中可以看出，两种处理方式的加速比都呈先增长后下降最后趋于稳定的趋势。这是因为前期数据包数量小，两系统内线程都有闲置情况，数据包被当前交换机快速处理完毕，因此前期吞吐率较大；随着数据包数量的增加，系统内空闲线程逐渐减少，线程利用率增大，吞吐率也随之增大，在 2～3 s 间两系统的吞吐率达到最大值；而随着数据包数量的持续增加，系统内数据包产生积压，在队列中排队等待处理，因此吞吐率开始下降并在某一值趋于稳定。在处理数据包的整个过程中，流水线并行算法的吞吐率都明显高于串行算法，前者的吞吐率约为后者的两倍，且流水线并行算法吞吐率的增长速率也明显高于串行算法。

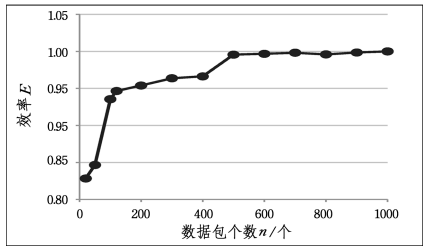

图 4 流水线系统效率变化图

Fig. 4 Efficiency change diagram of pipeline system

图 4 所示是流水线系统随要处理的数据包数的增加，系统效率 E 的变化示意图。可以看出，当前期数据包数量较少时，系统效率较低；随着数据包数量的增加，效率也逐渐提高；最后，当数据包达到一定数量后，系统效率稳定在 0.99 以上。这是因为上文中提到的流水线系统的处理时间包括通过时间、排空时间和额外开销所占时间。在前期即数据包数量较少时，系统整体的运行时间较少，流水线的通过、排空和额外开销时间所占比例较大，因此系统效率较低。随着数据包数量的增加，系统运行时间增长，系统流水段的处理时间所占比例越来越大，流水线的通过、排空和额外开销所占时间比例逐渐减小，效率也随之增大。最后，当数据包数量足够大，流水线系统满负荷运作时，系统效率趋于稳定，稳定值接近 100%。

从上述测试结果中不难看出，本系统能够明显提升交换机处理数据包的性能。

结束语 基于 SDN 控制与转发分离的事实，充分挖掘 SDN 控制与转发分离提供的内在并行性，可以提高 SDN 运行机制的性能。本文将 SDN 网络架构和流水线技术相结合，设计出 OpenFlow 交换机数据包流水线并发处理机制，并通过系统评测说明本系统对 SDN 网络中的数据包转发性能有显著提升。当数据包的发送数量巨大时，可采用本文的流水线机制交换机来提升转发速率。本系统为将 SDN 的思想运用到大规模互联网中，为后者性能的进一步优化提供了新的可能性。考虑流水线划分的多种可能性，权衡处理粒度与效率，进一步优化流水线处理，将是今后工作的重点。

参 考 文 献

[1] FOUNDATION O N. Software-Defined Networking: The New Norm for Networks [OL]. https://www.researchgate.net/publication/272829895_software-Difined_Networking_The_New_Norm-for-Networks.

[2] WANG X W, LI J, TAN Z H, et al. The State of the Art and Future Tendency of "Internet +" Oriented Network Technology [J]. Journal of Computer Research and Development, 2016, 53(4): 729-741. (in Chinese)
王兴伟，李婕，谭振华，等. 面向"互联网＋"的网络技术发展现状与未来趋势[J]. 计算机研究与发展，2016，53(4)：729-741.

[3] MEYER D. The Software-Defined-Networking Research Group [J]. IEEE Internet Computing,2013,17(6):84-87.

[4] BU C,WANG X,HUANG M,et al. SDNFV-based Dynamic Network Function Deployment: Model and Mechanism [J]. IEEE Communications Letters,2018,22(1):93-96.

[5] BU C,WANG X,CHENG H,et al. Enabling Adaptive Routing Service Customization via the Integration of SDN and NFV[J]. Journal of Network & Computer Applications,2017,93:123-136.

[6] LV J,WANG X,HUANG M,et al. RISC:ICN routing mechanism incorporating SDN and community division [J]. Computer Networks,2017,123:88-103.

[7] HELLER B,SHERWOOD R,MCKEOWN N. The controller placement problem [J]. Acm Sigcomm Computer Communication Review,2013,42(4):7-12.

[8] KOPONEN T,CASADO M,GUDE N,et al. Onix:a distributed control platform for large-scale production networks [C]// Usenix Conference on Operating Systems Design and Implementation. USENIX Association,2010:351-364.

[9] HASSAS YEGANEH S,GANJALI Y. Kandoo:a framework for efficient and scalable offloading of control applications [C]// The Workshop on Hot Topics in Software Defined Networks. ACM,2012:19-24.

[10] CURTIS A R,MOGUL J C,TOURRILHES J,et al. DevoFlow: Scaling flow management for high-performance networks [J]. Acm Sigcomm Computer Communication Review,2015,41(4):254-265.

[11] CURTIS A R,MOGUL J C,TOURRILHES J,et al. DevoFlow: Scaling flow management for high-performance networks [J]. Acm Sigcomm Computer Communication Review,2015,41(4):254-265.

[12] KANG N X,LIU Z M,JENNIFER R,et al. Optimizing the one big switch abstraction in software-defined networks[C]// ACM Conference on Emerging NETWORKING Experiments and Technologies. 2013:13-24.

[13] SYRIVELIS D, PARISIS G, TROSSEN D, et al. Pursuing a Software Defined Information-centric Network [C]// European Workshop on Software Defined Networking. 2012:103-108.

[14] BARATH R,MARTÍN C,TEEMU K,et al. Software-defined internet architecture: decoupling architecture from infrastructure [C]// ACM Workshop on Hot Topics in Networks. 2012:43-48.

[15] MCKEOWN N,ANDERSON T,BALAKRISHNAN H,et al. OpenFlow:enabling innovation in campus networks [J]. Acm Sigcomm Computer Communication Review,2008,38(2):69-74.

[16] Aaron. PIPELINING[EB/OL]. http://blog.163.com/sunshine_linting/blog/static/4489332320117250104945 4.

[17] OpenFlow Switch Specification, Version 1.1.0[EB/OL]. http://archive.openflow.org.

(上接第 294 页)

结束语　本文在 GPU 上实现了分子动力学模拟核心算法——Cell Verlet 算法,并设计了相应的模拟程序 Cell MD。通过实验分析,Cell MD 的算法实现具有可扩展性,相比于多核 CPU,其具有明显优势,能够达到当前主流 LAMMPS 软件同等的并行效率。后期,我们考虑将 Cell MD 实现扩展到多 GPU 环境,以进一步提升加速比。

参 考 文 献

[1] SCHLICK T,COLLEPARDOGUEVARA R,HALVORSEN L A,et al. Biomolecular modeling and simulation:a field coming of age[J]. Quarterly Reviews of Biophysics,2011,44(2):191.

[2] YUKO O. Molecular simulations in generalised ensemble[J]. Molecular Simulation,2012,38(14-15):1282-1296.

[3] BROWN W M,WANG P,PLIMPTON S J,et al. Implementing molecular dynamics on hybrid high performance computers-short range forces [J]. Computer Physics Communications,2011,182(4):898-911.

[4] ABRAHAM M J,MURTOLA T,SCHULZ R,et al. GROMACS:High performance molecular simulations through multi-level parallelism from laptops to supercomputers[J]. Softwarex,2015,1-2(C):19-25.

[5] VERLET L. Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules[J]. Health Physics,1967,22(1):79-85.

[6] MATTSON W,RICE B M. Near-neighbor calculations using a modified cell-linked list method[J]. Computer Physics Communications,1999,119(2-3):135-148.

[7] GLASER J,NGUYEN T D,ANDERSON J A,et al. Strong scaling of general-purpose molecular dynamics simulations on GPUs[J]. Computer Physics Communications,2015,192:97-107.

[8] FEI H,ZHANG Y Q,WANG K,et al. Parallel Algorithm and Implementation for Molecular Dynamics Simulation Based on GPU [J]. Computer Science,2011,38(9):276-278. (in Chinese)
费辉,张云泉,王可,等. 基于 GPU 的分子动力学模拟并行化及实现[J]. 计算机科学,2011,38(9):276-278.

[9] CHEN F G,GE W,LI J H. Implementation of Complex Multiphase Flow Molecular Dynamics Simulation on GPU [J]. Chinese Science (Series B:Chemistry),2008,38(12):1120-1128. (in Chinese)
陈飞国,葛蔚,李静海. 复杂多相流动分子动力学模拟在 GPU 上的实现[J]. 中国科学:B 辑,2008,38(12):1120-1128.

[10] PLIMPTON S. Fast parallel algorithms for short-range molecular dynamics[J]. Journal of Computational Physics,1995,117(1):1-19.

[11] PLIMPTON S,HENDRICKSON B. A new parallel method for molecular dynamics simulation of macromolecular systems[J]. Journal of Computational Chemistry,1994,17(3):326-337.

[12] GREST G S,DÜNWEG B,KREMER K. Vectorized link cell Fortran code for molecular dynamics simulations for a large number of particles[J]. Computer Physics Communications,1989,55(3):269-285.

[13] HILBERT D. Über die stetige Abbildungeiner Linie auf ein Flächenstück[M]// Dritter Band: Analysis · Grundlagen der Mathematik · Physik Verschiedenes. Berlin: Springer,1935,38(3):459-460.

[14] BUTZ A R. Alternative Algorithm for Hilbert's Space-Filling Curve[J]. IEEE Computer Society,1971,20(4):424-426.