doi:10.3772/j.issn.1002-0470.2023.02.001

紧耦合异构线程处理器①

李文青② 齐 寒 肖子原 朱威浦 王 剑③

(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)

(中国科学院大学计算机科学与技术学院 北京 100049)

摘 要 异构计算为系统达到更高的性能功耗比提供了新的思路和方向,但异构系统中中央处理器(CPU)和加速器协同执行任务的过程中大量的控制信号传输和数据搬运始终是系统性能的一个重要瓶颈。对此,本文提出了一种紧耦合异构线程处理器结构,包括一个硬件 CPU 线程和一个硬件加速器线程,二者采用流水线紧耦合的硬件线程间通信接口和共享存储的方式降低了通信代价,大幅提高了系统性能。为验证该结构的优势,本文在开源 BOOM 核的基础上设计了硬件线程间通信接口,实现了一个具有高级加密标准(AES)加速器的紧耦合异构线程处理器,并在现场可编程门阵列(FPGA)上进行了评估。结果显示,在加密任务中,该处理器吞吐量约是 Intel Comet Lake 使用 AES 指令集(AES-NI)的 5.7 倍,是 BOOM 平台上仅使用通用指令的 4000 倍。实验进一步验证了通过 CPU 和加速器快速通信实现的细粒度并行可以取得更多的性能收益。由此得出结论:该结构能敏捷地将加速器整合到 CPU 周围,有效降低了通信时间,实现 CPU 线程和加速器线程的细粒度并行,有效地发挥出异构计算的优势,取得可观的性能收益。

关键词 异构计算; 异构接口; 紧耦合; 通信; 细粒度并行

0 引 言

为了追求更高的性能功耗比,或满足特定领域性能、功耗的要求,越来越多的专用硬件加速器出现在智能移动设备、用户终端和计算中心。例如,麻省理工学院研发了面向智能移动设备深度学习的 Eyeriss 系列加速器[1-3]。NVIDIA 公司为了满足用户游戏、视频、工作学习等需求,为个人计算机系统设计的图形加速器(graphics processing unit,GPU)[4]。Google 公司为加速数据中心的神经网络推理和训练过程而研发的张量处理单元(tensor processing unit,TPU)系列芯片[5-6]。在信号处理[7]、人工智能[8-9]、图形图像[10]、高性能计算[11-12]等众多领域,中央处理器(central processing unit, CPU)加硬件加速器的异构计算模式都有着更突出的表现,受到了学术界和工业界的共同关注。

异构计算系统针对目标应用设计硬件加速器,使用专用的硬件计算部件替代了软件使用 CPU 完成的大量工作,对整体性能有大幅提升。加速器的设计主要包括硬件加速计算部分和与 CPU 通信部分。计算部分根据特定应用设计专用硬件电路,快速完成特定操作。而对于与 CPU 通信部份,由于原本全部是软件使用 CPU 做的工作,现在一部分由硬件加速器完成,所以加速器需要设计通信和控制电路,利用它发送或接收控制、请求信号并与 CPU 进行数据交换。CPU 与加速器进行必要的通信,从而相互配合共同完成任务。

CPU 和加速器的通信速度是异构计算系统的

① 中国科学院战略性先导科技专项(XDC05020100)资助项目。

② 女,1994 年生,博士生;研究方向:计算机系统结构,处理器设计;E-mail:liwenqing@ict.ac.cn。

③ 通信作者,E-mail: jw@ict.ac.cn。

(收稿日期:2021-11-09)

重要性能瓶颈,也成为研究的重点方向。在异构系统运行过程中,为了使用加速器,CPU 需要对加速器进行必要的配置、启动、停止等控制,加速器也可能产生中断、例外等请求交给 CPU 处理。加速器计算的输入数据、中间结果和输出数据等都会产生数据搬运的开销。文献[13]提出异构系统中存在“交互墙”,并指出算法复杂度增长的同时将会产生更多交互,而且交互速度的提升低于加速器计算速度的提升。文献[1-3,14]也指出了异构结构中数据搬运产生的性能功耗损失严重。

为了降低通信开销,更有效地发挥异构计算的优势,本文提出了一种紧耦合异构线程处理器结构。该结构在通信方面代价极小。具体而言,CPU 采用流水线紧耦合的方式发送控制信息到加速器,通过寄存器进行信息交换,并且使用共享缓存(cache)的方式减少数据搬运开销。该结构的优势在于,CPU 和加速器在不同的线程上工作,通过快速地通信实现更好的协同,挖掘更细粒度任务并行,从而提升系统整体性能。本文贡献主要有以下几个方面。

(1)提出了一种紧耦合异构线程处理器结构,在解耦 CPU 和加速器的同时大幅降低了二者的通信代价,加强了 CPU 和加速器的协同合作,实现细粒度并行,从而提升了整体性能。

(2)依据本文提出的结构实现了一个具有加解密硬件加速器的处理器,并通过 Design Compiler 综合工具和现场可编程门阵列(field programmable gate array, FPGA)平台进行了面积、功耗和性能评估。

(3)通过实验和分析验证了本文所提结构的有效性。实验对比结果表明,本文实现的处理器完成加密任务吞吐量约是使用 Intel 高级加密标准指令集(advanced encryption standard new instructions, AES-NI)的 5.7 倍,是纯软件吞吐量的 4000 多倍,同时验证分析了通过 CPU 和加速器快速通信实现的细粒度并行可以取得更多的性能收益。

1 背景和相关工作

1.1 硬件多线程架构

为了使处理器核中的计算资源得到有效利用,通过挖掘并行性提升系统性能,硬件多线程架构在用于研究和商业的各类处理器设计中被广泛使用。硬件多线程处理器在没有软件干预的情况下有多条流水线同时互不干扰地执行。这种架构相较于单一流水线架构减少了软件必要的上下文切换,并通过挖掘线程级并行补充指令级并行的性能不足。

自二十世纪五六十年代开始,DYSEA[15]、CDC6600[16]计算机就使用了硬件多线程的思想隐藏 IO 操作的延迟。Horizon[17]实现了 CPU 核内细粒度多线程,使用超长指令字(very long instruction word,VLIW),但依旧没有突破指令级并行的限制,并且严重依赖于程序行为和编译优化。Sparcle[18]研究了单芯片多处理器核(chip multiprocessors, CMP)共享内存核和地址空间方法的硬件多线程架构,并指出由于通信而不可避免的延迟及缓解方法。Intel Pentium 4[19]将同时多线程(simultaneous multithreading, SMT)用于商业级通用 CPU 中,共享计算部件以提高利用率从而提升性能。再之后服务器和桌面计算机的 CPU 大量使用 CMP 和 SMT 组合的处理器,以提高系统性能。

CMP 的方式便于处理器核之间的互联,如今也多用于连接异构核,使用领域专用加速器提升目标应用性能,但其软硬件通信开销也是高昂的[20-21]。SMT 的方式共用计算部件没有通信开销,对于通用处理器擅长处理的事务管理等性能提升效果明显,但不适合使用这种方式添加专用硬件对新兴应用加速,传统的处理器结构也会对添加的异构计算部件带来性能上的制约[22-23]。

本文提出的紧耦合异构线程处理器,采用紧耦合的方式消除结构和软件上带来的通信开销。异构计算部件和 CPU 处于不同的硬件线程,如 CMP 方式不共享内部寄存器与计算单元,但共享地址空间,拥有独立的执行流,发挥专用硬件的算力。

1.2 异构处理器结构

由于加速器的规模和需求不同,异构系统中 CPU 和加速器的互连方式也有所不同。按照两者的连接方式可以分为片间连接和片上连接,片上连接通常是通过片上总线连接或是流水线紧耦合。

片间连接的加速器是作为独立的加速卡连接到

处理器上的,这类加速器在设计上较为灵活,例如GPU、TPU、DianNao[24]、ESE[25]等。它们都拥有本地的存储单元、较为复杂的控制逻辑,通过高速串行计算机扩展总线标准(peripheral component interconnect express, PCIe)等总线连接。这样的加速器在设计上比较独立和自由,但软硬件设计复杂。CPU和加速器在控制信号的通信上需要驱动、内核等支持,因此有着较高的延迟。数据需要在内存和加速器本地存储之间搬运,延迟较高。这种加速器通常会通过数据复用、压缩、剪枝等方法缓解通信开销大的问题。

面对频繁出现需要更多通信的操作,近年来将加速器集成到片上通过系统总线与CPU进行连接的方式被越来越多地使用。例如APPLE M1 CPU与片上加速器基于统一的存储体系,IBM POWER9和IBM z15片上集成数据压缩加速器[26], AMD将GPU集成到片上的加速处理器(accelerated processing unit, APU)。比起片间连接,这种连接方式拉近了加速器和CPU的距离,使得控制信号传输得更快,并且可以使用共享存储的方式减少数据通信。但是由于复杂的软件栈存在,控制信号的传输速度依然不能支持加速器和CPU过于频繁地通信。

图1(a)为总线方式连接的CPU和加速器执行示意图,左边为CPU执行过程,右边为加速器执行过程,阴影部分为CPU与加速器交互或是加速器正在执行,其余部分为停止或执行其他任务。CPU启动加速器需要进行较多条指令用于准备、配置和数据搬运,然后CPU空闲或者切换到其他任务,加速器开始执行,在执行过程中如果需要和CPU进行交互,又需要较长时间的通信。

为了追求更高的性能,各大处理器厂商在基础指令集的基础上都添加了专用扩展指令对特有的场景进行硬件加速,最为普遍的是处理器内部的向量部件、浮点协处理器等。紧密集成到处理器流水线中的指令集扩展加速器通过共享现有处理器资源来减少通信开销,达到了较好的性能。但这些硬件加速器都是以特有的方式连接到CPU中,与处理器流水线的紧耦合使得工程实现细节上也有着密切的联系,这种方式添加硬件加速器门槛较高,需要对CPU实现细节十分熟悉,并且验证和实现会产生大量的工程成本。

文献[27-30]研究在通用CPU上添加流水线紧耦合的加速器以达到更高的单核性能。但这些工作使用的处理器比较简单或加速器设计不够灵活独立,现代CPU大多是超标量乱序的结构,它们对于较为复杂的处理器核不一定适用。文献[31]指出传统处理器架构对计算部件利用率有影响,这种情况添加计算部件也会受到处理器架构的限制。

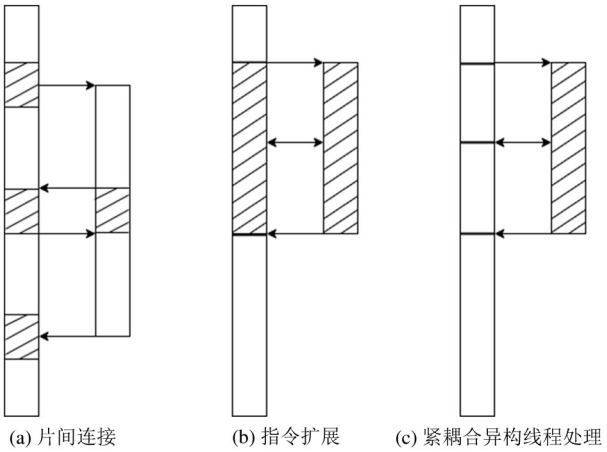

图1　不同连接方式下CPU和加速器执行过程

传统指令集扩展的硬件加速器受限于指令集并行能带来的收益,加速器功能不能过于复杂,否则会影响到处理器本身的频率和性能。如图1(b)所示,如果指令过于复杂、执行时间较长,则使得这条指令在重排序缓存(reorder buffer, ROB)中不能退出,从而阻塞流水线;如果功能适当,不会阻塞流水线,但是能做的操作有限,需要许多条新添加指令及辅助指令共同完成任务,CPU依然会花费时间在这些指令从取指开始的各个流水线阶段。

本文设计的异构线程处理器执行过程如图1(c)所示,通过共享存储减少了数据通信时间,通过1条或几条指令即可与加速器进行一次控制信号的通信。启动加速器之后CPU即可与加速器并行执行其他任务,执行过程中CPU和加速器依然是1条或者几条指令即可完成线程间通信。

2　整体设计

为了更好地满足应用的需求,提升系统整体的

性能，本文提出了一种紧耦合异构线程处理器结构。针对不同的应用领域，使用本文设计的结构可以更敏捷地将计算专用的硬件电路快速整合到通用处理器周围，采用紧耦合的结构最小化通信时间，实现处理器事务处理和计算处于不同硬件线程并行执行的同时，仍然能够快速通信、协同工作，并且挖掘更细粒度并行的潜力。

2.1　紧耦合异构线程处理器结构

本文提出的紧耦合异构线程处理器结构如图 2 所示，由 CPU 硬件线程和加速器硬件线程紧耦合组合成处理器核心。CPU pipeline 部分是通用 CPU 的流水线逻辑和资源，负责核心态和事务处理，构成 CPU 硬件线程（图 2 中的 hardware thread A）；Accelerator 是为目标应用设计的硬件加速单元，受 CPU 线程调度，工作在用户态，与 CPU 共享 cache，构成加速器硬件线程（图 2 中的 hardware thread B）。2 个硬件线程运行不同的软件线程，且硬件电路之间不直接连接，通过中间部分的部件进行线程交互和数据访问。图中异构接口（YiGouJieKou，YGJK）部分是为支持这种结构设计的 CPU 和加速器的接口，与 CPU 流水线和访存子系统紧密连接，并向加速器提供统一的控制和访存端口。

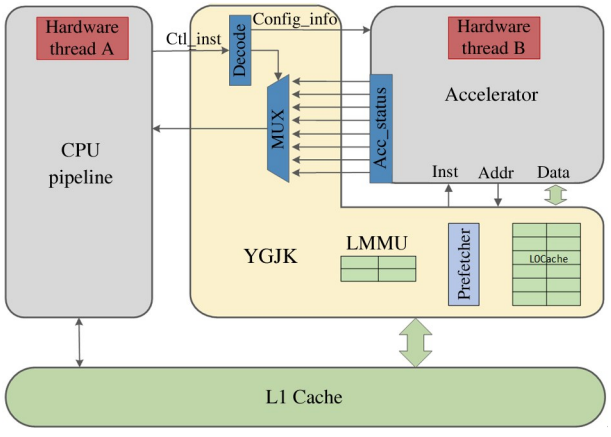

图 2　紧耦合异构线程处理器结构

2.2　设计解耦

通过在 CPU 和硬件计算部件之间提供统一的接口，将通用处理器和紧耦合加速器进行解耦。YGJK 接收来自 CPU 的指令再转成配置或控制信息传到加速器并将加速器的信息返回给 CPU。同时 YGJK 接管加速器的访存，由 YGJK 处理加速器的读写请求并和 cache 进行交互，YGJK 内设有缓存和预取等机制可以对访存进一步优化。

由于 YGJK 的存在，CPU 和硬件加速器互相透明。CPU 将 YGJK 当做一个功能部件，添加发往 YGJK 的指令信息、操作数等通路，将控制信息发往 YGJK。对于访存的支持，L1 Cache 添加一个端口，用来处理 YGJK 的访存请求。加速器接收来自 YGJK 的控制信号和配置信息并把查询的信息或者加速器状态等信息发往 YGJK。加速器的访存直接和 YGJK 进行交互，发出读写请求并等待或者发出数据。

这样，加速器设计实现上可以更加独立。加速器只需要满足控制信号和访存传输的规则，内部逻辑实现自由，设计人员不需要对处理器细节有很深入的了解，也不会由于加速器的添加为 CPU 引入新的功能错误，节省了传统设计中因添加新的硬件产生的大量 CPU 验证工程成本。CPU 的可扩展性因此得到了提升，并且降低了加速器紧耦合到 CPU 流水线中的门槛和成本。

2.3　异构并行

利用本文提出的在设计上解耦的处理器结构，相关的指令在执行上是异构多线程的。用户态可控制的加速器仅需要 1 条或几条指令就可以配置启动，之后 CPU 和加速器可以处于不同的线程中并行执行各自的操作，在需要通信的时候通过 CPU 指令和 YGJK 中的寄存器进行交互。

为实现硬件线程间的通信，本文设计了 3 类指令，分别是加速器启动指令 launch 和跨线程的寄存器间数据移动指令 mta（move to accelerator）、mfa（move from accelerator）。其中，launch 指令在 CPU 中译码后发往 YGJK，通过 YGJK 二次译码 function 域实现不同加速功能的启动。mta 指令完成数据从 CPU 中寄存器到加速器寄存器的搬运，具有 2 个源操作数，每次可以移动 2 个寄存器数值到加速器中，加速器按照顺序接收。mfa 指令有一个目的寄存器，根据指令中 function 域在 YGJK 中译码，并选择对应的加速器状态寄存器的值返回给指令，存入目的寄存器。所有发往 YGJK 的指令都立刻退出，不

会因为等待加速器的执行对流水线造成阻塞。

同时为了支持处理器多线程工作,在 YGJK 中添加内存管理单元(local memory management unit, LMMU)支持,加速器的访存请求在 YGJK 的 LMMU 中就可以进行虚实地址转换,然后发送到对应 cache,不会影响到 CPU 的访存流水线。

流水线紧耦合的方式使得加速器指令到达加速器仅需几个时钟周期,加速器的信息通过寄存器传递给 CPU。这种代价极小的通信方式允许 CPU 和加速器在短时间内进行较多的交互,加速器可以以较小的任务粒度频繁启动,或是在执行过程中与 CPU 频繁交互。细粒度任务级并行的方式突破了指令级并行的限制,又达到了传统任务级并行不能完成的细粒度。

2.4 共享存储

访存速度对于现代计算机而言是很大的性能瓶颈,减少访存次数和添加预取等优化手段都可以大幅提高处理器整体性能。在加速器具有独立存储空间的异构计算系统中,数据搬运相关的工作占据了计算任务很大比例。

CPU 和加速器采用共享存储的方式减少数据通信的开销,减少了输入、输出及中间结果在 CPU 和加速器存储空间之间的搬运。YGJK 为加速器提供访存请求的处理功能,通过将 YGJK 的访存端口连接到 cache 上并在 YGJK 中添加新的 L0 Cache 的方式,为加速器缓存计算所需数据。考虑到加速器目标应用访问的数据通常是地址连续的,将 YGJK 到 cache 的访存通路增加至 1 个 cache 行大小,从而减少了访存次数。

2.5 软件接口

用户程序在需要使用加速器前,使用专门的系统调用获取加速器使用权,对 YGJK 中的 LMMU 地址空间进行配置,完成加速器的初始化。

后续操作和控制均在用户态完成,因此不需要设计加速器驱动程序。在用户态程序中选择使用相应的线程通信指令对加速器进行控制。将需要传入加速器的配置参数、描述符、控制信号存入 mta 指令的源寄存器中,然后使用 mta 指令按顺序发往 YGJK。YGJK 读取源寄存器并存入加速器配置寄存器中,加速器获取计算所需的信息后,CPU 发送 launch 指令,加速器即可开始相应的计算任务,CPU 也可以去执行其他指令。在计算任务进行中,CPU 需要查询加速器状态时,发送对应的 mfa 指令,YGJK 会选择状态寄存器中相应的值返回,存入 mfa 指令目的寄存器中。这意味着对加速器的控制仅需 1 条或几条指令就可以完成。

软件接口较为通用灵活,修改更换加速器后处理器流水线和指令不需要改动。通过设计统一的专用指令发往 YGJK,YGJK 再根据指令进行简单的二次译码,需要传到加速器的控制信号和寄存器值则传入加速器,而较为通用的功能在 YGJK 中完成。加速器设计时定义好接收参数的方式和查询加速器状态的指令码和操作数,软件使用时仅需按照加速器定义使用对应的指令和操作数就可以操纵加速器,对于指令通路和 CPU 流水线无需再进行任何修改。

3 异构线程处理器实现

为验证本文提出的结构的合理性和有效性,本节对该结构进行了一种实现。选择一个开源的乱序多发射处理器核 BOOM(Berkeley out-of-order machine)[32],对其添加专用接口来支持加速模块和 CPU 之间的解耦与线程间通信,实现了一个高级加密标准(advanced encryption standard, AES)模块作为硬件加速器部分。

3.1 BOOM 处理器核

BOOM 是一款开源 RISC-V 指令集架构的开源处理器核。BOOM 使用硬件构造语言 Chisel 实现,是可综合、参数化的超标量、乱序处理器核。因为 BOOM 具有现代 CPU 的大部分特征,并且加速器应用对 CPU 控制信号和访存很敏感,所以选择使用 BOOM 更具有研究意义。

为了支持本文提出的结构,本文对 BOOM 处理器访存子系统进行了一些修改。在访存子系统中添加了对 YGJK 访存请求的处理,一级数据缓存(level1 data cache, L1 Dcache)为 YGJK 提供了整行数据读写。图 3 所示是 L1 Dcache 的工作流程,array 是

L1 Dcache 的存储单元；s0 为 Dcache 收到的请求信号，s1 ~ s5 表示 Dcache 在不同阶段对信号及处理结果进行的寄存器缓存；圆角矩形是处理逻辑；Whole line 和 Line buffer 分别存储 Dcache 中读出和请求写入的整行数据。L1 Dcache 接收到访存请求判断是否来自 YGJK，分别在不同阶段对读写请求进行处理，以保证正确实现 LSU（load store unit）和 YGJK 对 L1 Dcache array 的整行读写。

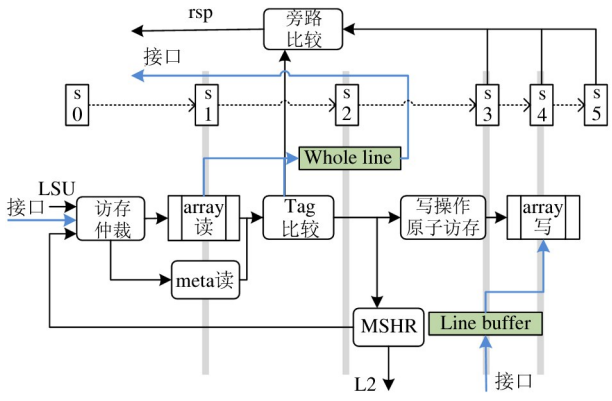

图 3　Dcache 工作流程图

3.2　异构接口

YGJK 是专门负责 CPU 线程和加速器线程交互的部件，YGJK 与处理器流水线连接，接收来自 CPU 流水线中的指令和操作数，经 YGJK 二次译码发往加速器并做对应操作。YGJK 与加速器状态寄存器相连接，可以直接返回对应寄存器值给来自 CPU 的查询指令。

为了支持 2 个线程独立执行，YGJK 内设有 LMMU 供加速器访存时使用，访存请求通过 YGJK 发往 cache。YGJK 内有 cache 为加速器提供数据缓存，并实现硬件预取机制对访存进行优化。

3.3　AES 加密模块

AES 加密广泛应用于主流加密算法中，AES 算法大多数操作为移位、查表和异或，软件实现需要很多条指令才能完成，但硬件操作比较简单。各硬件厂商出于对加密算法的加速以及基于硬件设计的安全功能，在处理器中添加 AES 硬件加速单元。如 Intel 通过指令集扩展提供 AES 专用指令、IBM power 系列设计 AES 加密模块[33]、ARM 的 CryptoCell 安全 IP 核。

为验证本文提出结构的有效性，使用 Chisel 语言编写了 AES 加速器作为例子，通过 YGJK 连接到 BOOM 中。加速器结构如图 4 所示，config/ctl 部分接收来自 YGJK 的信号；status 存储加速器需要传到 YGJK 的实时信息。该加速器支持 128 位、192 位和 256 位加密模式，对明文加密每一轮需要 1 个时钟周期。加速器在接收到加密模式、密钥、明文长度、明文起始地址和密文起始地址等配置信息后开始工作，向 YGJK 发送读明文和写密文请求，并且将加密进度等信息实时反馈到 YGJK 中。

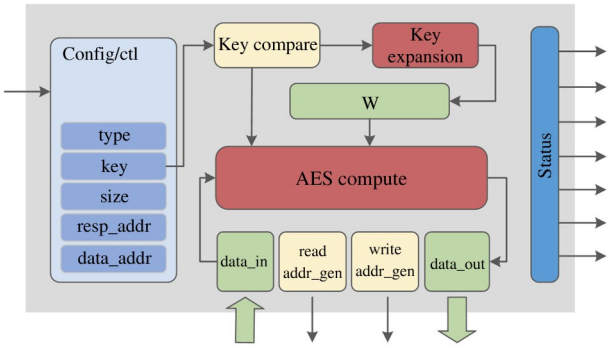

图 4　AES 加解密硬件加速器

4　实　验

第 3 节实现的多线程处理器的实验平台为 Xilinx Virtex-7 FPGA VC 709 开发板，测试程序为 C 语言编写，使用 RISC-V 64 位交叉编译工具 riscv64-unknown-elf-gcc 8.3.0。4.3 节对比平台的处理器为 Intel Comet Lake，编译器为 gcc 7.5.0。

4.1　时序面积功耗

使用 Synopsys 的 Design Compiler 在 tsmc 45 nm 工艺库下对本文实现的紧耦合多线程处理器进行了评估，结果如表 1 所示。

表 1　处理器 Design Compiler 综合结果

	关键路径延迟/ns	面积/μm^2	功耗/W
BOOM	0.99	4 752 909	4.7007
BOOM + YGJK	0.99	4 886 281	4.7924
比率	0	2.81%	1.95%

根据综合结果计算，相较于原生的 BOOM，添加 YGJK，逻辑门面积增长仅为 2.81%，功耗增长仅

1.95%。实现 YGJK 并不需要占用很多片上资源,并且这种方式添加的加速器因为连接方式简单,需要处理的交互方面的逻辑少,大幅减少了面积和功耗的增长。

实验中,对添加 AES 加速器的 BOOM 进行综合,查看电路关键路径。在 1 GHz 主频下,添加加速器前后的处理器都可以满足主频要求且关键路径延迟不变。查看添加的 YGJK 以及加速器的关键路径延迟为 0.85 ns,小于处理器整体关键路径,说明不会因为添加的 YGJK 和加速器对原来的处理器频率造成影响。因此,后续实验中使用了时钟周期数表示处理器性能。

4.2 延迟带宽

为了获得加速器访存的延迟,本实验中实现了一个功能简单的模块充当加速器。这个模块只是简单地发送读请求和地址到 YGJK 请求数据,并且使用寄存器记录从发出请求到收到数据的时钟周期数。编写 C 程序保证先将数据放入 L1 Dcache,再启动加速器。

根据实验运行结果,在没有任何阻塞的情况下,YGJK 发出 L1 Dcache 可以命中的读数请求,从 YGJK 发出请求到收到 512 bits 数据占用了 3 个时钟周期。同样的无阻塞情况下,加速器发出读数据请求到收到 512 bits 数据,YGJK 的 cache 如果没有命中,所占用时间为 5 个时钟周期;YGJK 中的 cache 命中,仅需 1 个时钟周期加速器即可获取数据。

本文这种结构的 CPU 与加速器耦合度很高,数据通信时间短。加速器请求到 YGJK 再到 L1 Dcache 的时钟周期数与 BOOM 访存单元以及现代 CPU 访存时间相当,并拥有更宽的数据通路。

4.3 吞吐量

本节将带有 AES 硬件加速器的 BOOM 运行在 FPGA 平台上,编写 C 程序分别使用硬件加速器和仅使用 BOOM 进行明文加密实验,并与使用 Intel AES 指令完成相同加密任务进行比较。

测试程序伪代码如代码 1 所示。1 ~ 3 行将 AES 加速器需要的配置参数放入寄存器中,分别是加密类型(密钥位数,128 bits、192 bits 或 256 bits)、明文长度、密钥高位、密钥低位(测试中使用的 128 bits 密钥)、密文地址和加密后存储地址。5 ~ 8 行使用 mta 指令将配置参数传到加速器并使用 launch 指令进行启动。随后 CPU 可以执行其他任务,或是 11 行使用 mfa 指令查询加速器运行状态等待加速器任务结束,加速器线程停止,CPU 线程继续完成其他工作。可以看出,配置和启动加速器仅需要这几条配置指令运行结束,无阻塞的指令从发出到提交只需十几个时钟周期,远少于计算时间。

代码 1 AES 加速器编程模型

```
1    r0 = AESType;      r1 = SIZE;
2    r2 = key1;         r3 = key2;
3    r4 = dataAddr;     r5 = destAddr;
4    //配置启动加速器
5    MTA(r0, r1)
6    MTA(r2, r3)
7    MTA(r4, r5)
8    LAUNCH()
9    Do something else;
10   //查询等待加速器结束
11   while(! MFA0());
12   Do something else;
```

图 5 展示了这 3 种方式在 128 位加密模式下对 512 B ~ 4 MB 明文段进行加密处理的性能,纵坐标是平均加密每字节需要的时钟周期数,所以数值越低性能越好。可以看出,使用紧耦合异构线程处理器结构有着更好的性能,吞吐量约是 Intel AES 指令的 5.7 倍,是纯软件实现的 4000 多倍。

AES 算法规律地重复着一个数组内的移位、异或和查表。纯软件实现 AES 加密操作需要在一个数组中反复地移动、交换、计算其中的 1 个或 2 个元素,虽然同一列或者同一行元素操作互相无关,但是也无法并行执行,而且整个过程还需要许多其他指令和寄存器的辅助才可以完成。但是硬件实现却非常容易,能并行完成这些操作,且不需要多余指令的辅助,因此纯软件实现完成加密任务所花费的时间是硬件加速的百倍甚至千倍。同时由于 AES 算法被广泛使用,这使得各大处理器厂商将 AES 操作做成特有的硬件模块集成在处理器中以获得更好的性能。

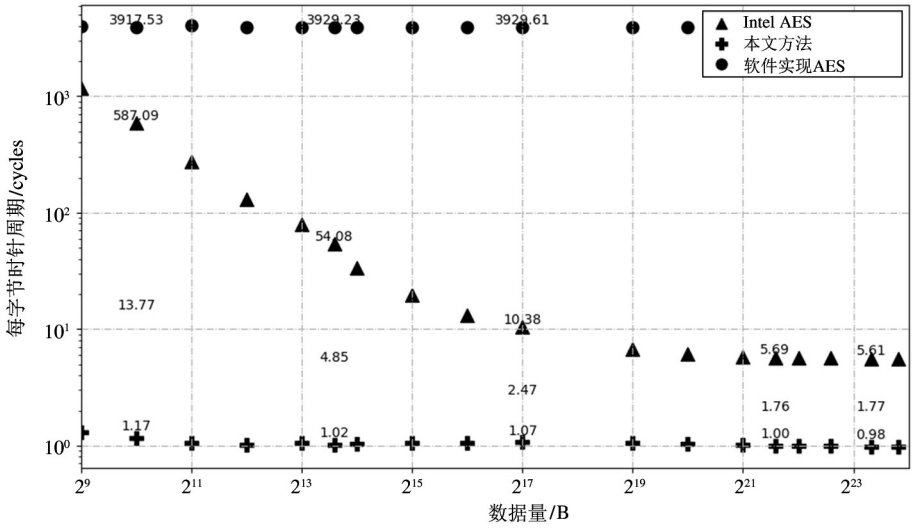

图 5　对 512 B 到 4 MB 明文段进行加密

Intel 设计了面向 AES 加密的专用指令,在加密开始阶段,需要先进行密钥扩展操作,并且由于 cache 中没有明文数据需要到内存中取数,当程序运行一段时间后 Intel 处理器对于这种流式访存行为会开启访存预取机制使得访存时间变短,因此图中三角形点(▲)随着明文长度增大每字节时钟周期数下降,后因启动开销占比变少、预取机制稳定后逐渐到达拐点保持稳定的加密吞吐量。

Intel AES 每条指令完成加密的一轮操作,因此完成整个任务依然需要指令来完成循环和数据搬运的操作。由于使用异构多线程结构,相较于传统指令集扩展方式硬件加速的粒度可以增大,并且减少了辅助指令,减轻了 CPU 负担,本文实现的 AES 加速器更有效地使用了硬件加速模块,可以达到更高的性能。

4.4　通信

紧耦合异构线程处理器将加速器和 CPU 处于 2 个不同线程运行的同时,可以利用通信速度快的优势,增加 CPU 和加速器的配合,挖掘更细粒度的并行。

CPU 和加速器配合的测试程序伪代码如代码 2 所示。同代码 1,1 ~ 8 行将加速器参数配置完成并启动,第 10 行使用 mfa 指令中读取加速器运行状态的指令 MFA0 获取加速器是否运行结束,如果没有结束就执行 while 中的代码;11 ~ 14 行中,通过使用 mfa 指令 MFA1 读取加速器正在计算的明文地址,提前将下一个需要计算的 cache 行预取到 L1 Dcache 中。查询指令也和其他跨线程指令一样无阻塞,所以执行速度很快,传输回来的数据具有实时性,可以依次进行预取。

代码 2　CPU 和加速器协同工作示例

```
1    r0 = AESType;    r1 = SIZE;
2    r2 = key1;       r3 = key2;
3    r4 = dataAddr;   r5 = destAddr;
4    //配置启动加速器
5    MTA(r0, r1)
6    MTA(r2, r3)
7    MTA(r4, r5)
8    LAUNCH()
9    //查询加速器状态协同工作
10   while(! MFA0()){
11   rd = MFA1()
12   ld rd + 64;
13   ld rd + 128;
14   …
15   }
16   Do something else;
```

本实验对从 512 B ~ 4 MB 明文段进行 128 位加密模式的加密计算。测试程序有 2 种,一种仅使用加速器完成加密工作,CPU 只负责配置启动加速器;另一种测试程序中加速器在加密计算过程中通过与 CPU 通信,根据加速器的加密进度,CPU 负责

提前准备加速器需要的明文数据。这 2 种测试程序分别运行在 BOOM 访存子系统打开硬件预取和关闭硬件预取的情况下,实验结果如图 6 所示,纵坐标是平均加密每字节需要的时钟周期数,所以数值越低性能越好。

图 6 中圆形的点(●)是没有硬件预取也没有软件预取的结果,加速器每次访存请求都不会在 cache 中命中,需要访问内存,所以整体性能是最差的。三角形的点(▲)是有硬件预取但是没有软件预取,相较于圆形点的性能有了一定的提升。因为流式的访存行为在 BOOM 访存子系统中的硬件预取机制比较容易能预测到将要访问的数据地址,在新的访存请求到达之前预取器已经到内存中取数据,这使得访存时间变短,整体性能就有了提升。使用硬件预取也是现代 CPU 对访存进行优化的一个重要且有效的手段。

X 形的点(×)是没有硬件预取只有软件预取的结果,通过依靠 CPU 和加速器之间的通信,CPU 线程根据加速器传来的信息准确进行接下来加速器访存的提前操作,为加速器提前准备数据。在加速器请求发出时,CPU 已经将数据准备好放在 L1 Dcache 中或者在访存过程中,缩短加速器访存时间。这种方式比仅有硬件预取的结果更好,硬件预取到的数存在 L2 cache,比软件预取存入 L1 Dcache 对于加速器来说访问延迟更高。并且硬件预取是硬件面向大多数应用程序进行了硬件优化机制;而软件预取是通过与加速器通信得到的准确信息,能在更合适的时间灵活配合,更能满足加速器的访存需求。得益于紧耦合结构极短的通信时间,加速器和 CPU 能做到传统异构多线程系统无法做到的细粒度并行。

正方形的点(□)是硬件预取和软件预取都存在的情况下的性能结果,可以看出,性能没有比仅有软件预取更好。硬件预取是处理器访存子系统的一个硬件部件,是对程序访存行为进了行预测,而本实验中的软件预取已经可以精确拿到将来需要的数据地址,所以增加了硬件预取并不能获得更多的性能收益。

从以上结果可以看出,使用 YGJK 支持的紧耦合异构线程处理器由于轻量、极快的通信,2 个线程之间发送的消息开销小、具有实时性,因此可以实现频繁通信、细粒度并行、精准协同合作,进一步挖掘性能潜力。

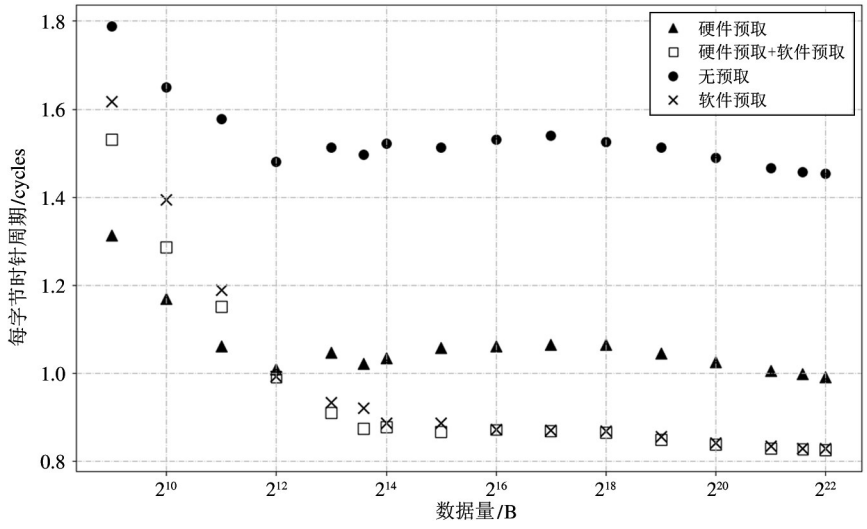

图 6　CPU 与加速器协同工作

5　结　论

在摩尔定律逐渐趋于终结的时候,异构计算在多个领域中展示出突出的表现。针对异构计算中通信开销大造成的性能瓶颈,本文提出了一种紧耦合异构线程处理器,极大降低了通信的代价,挖掘传统异构系统无法达到的细粒度并行,更好地协同合作提升系统性能。该结构同时也保持了 CPU 和加速

器的独立性、灵活性和软件接口的通用性。

通过对所提出结构的一种实现,验证了该结构的有效可行。实验结果表明,本文实现的处理器完成明文加密任务吞吐量约是 Intel 处理器使用 AES 指令吞吐量的 5.7 倍,是纯软件吞吐量的 4000 倍,并且实验验证了通过 CPU 和加速器快速通信实现的细粒度并行可以取得更多的性能收益。

参考文献

[1] CHEN Y H, EMER J, SZE V. Eyeriss: a spatial architecture for energy-efficient dataflow for convolutional neural networks[C]//The 43rd Annual International Symposium on Computer Architecture. Seoul: IEEE, 2016: 367-379.

[2] CHEN Y H, KRISHNA T, EMER J S, et al. Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks[C]//IEEE International Solid-state Circuits Conference. San Francisco: IEEE, 2016: 127-138.

[3] CHEN Y H, YANG T J, EMER J S, et al. Eyeriss v2: a flexible accelerator for emerging deep neural networks on mobile devices[J]. IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 2019,9(2):292-308.

[4] CHOQUETTE J, GANDHI W. NVIDIA A100 GPU: performance & innovation for GPU computing[C]//2020 IEEE Hot Chips 32 Symposium. Palo Alto: IEEE, 2020: 1-43.

[5] JOUPPI N P, YOUNG C, PATIL N, et al. In-datacenter performance analysis of a tensor processing unit[C]//Proceedings of the 44th Annual International Symposium on Computer Architecture. Toronto: IEEE, 2017: 1-12.

[6] JOUPPI N P, YOON D H, KURIAN G, et al. A domain-specific supercomputer for training deep neural networks [J]. Communications of the ACM, 2020, 63(7):67-78.

[7] 雷元武,陈小文,彭元喜. DSP 芯片中的高能效 FFT 加速器[J]. 计算机研究与发展, 2016, 53(7):1438-1446.

[8] 张士长, 王郁杰, 肖航, 等. 支持 CNN 与 LSTM 的二值权重神经网络芯片[J]. 高技术通讯, 2021, 31(2):122-128.

[9] 周聖元, 杜子东, 陈云霁. 稀疏神经网络加速器设计[J]. 高技术通讯, 2019, 29(3): 222-231.

[10] 张立志, 赵士彭, 赵皓宇, 等. 高性能 GPU 模拟器的实现[J]. 高技术通讯, 2020, 30(6):553-560.

[11] DONGARRA J. Emerging heterogeneous technologies for high performance computing[C]//The 22nd International Heterogeneity in Computing Workshop. Boston:IEEE,2013.

[12] EL B D, NGUYEN T T, JOURJON G, et al. HPC applications deployment on distributed heterogeneous computing platforms via OMF, OML and P2PDC[C]//2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing. Torino: IEEE, 2014: 617-623.

[13] DU Z D, GUO Q, ZHAO Y W, et al. Breaking the interaction wall: a DLPU-centric deep learning computing system[J]. IEEE Transactions on Computers, 2020, 71(1):209-222.

[14] FEINBERG B, HEYMAN B C, MIKHAILENKO D, et al. Commutative data reordering: a new technique to reduce data movement energy on sparse inference workloads[C]//2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture. Valencia: IEEE, 2020: 1076-1088.

[15] LEINER A L. System specifications for the DYSEAC[J]. Journal of the ACM, 1954,1(2): 57-81.

[16] THORNTON J E. Design of a computer: the control data 6600[M]. Glenview: Scott Foresman Company,1970.

[17] KUEHN J T, SMITH B J. The horizon supercomputing system: architecture and software[C]//Conference on High Performance Networking and Computing: Proceedings of the 1988 ACM/IEEE Conference on Supercomputing. Orlando: IEEE, 1988: 28-34.

[18] AGARWAL A . The MIT Alewife machine: architecture and performance[C]//Proceedings 22nd Annual International Symposium on Computer Architecture. Madison: IEEE, 2005: 2-13.

[19] HINTON G, SAGER D, UPTON M, et al. The microarchitecture of the Pentium© 4 processor[J]. Intel Technology Journal, 2001,1:1-13.

[20] WANG P H, COLLINS J D, CHINYA G N, et al. EXOCHI: architecture and programming environment for a heterogeneous multi-core multithreaded system[C]//Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation. San Diego: Association for Computing Machinery,2007:156-166.

[21] COTA E G, MANTOVANI P, GUGLIELMO G D, et al. An analysis of accelerator coupling in heterogeneous architectures[C]//The 52nd ACM/EDAC/IEEE Design Automation Conference. San Francisco: IEEE, 2015:1-6.

[22] NORI A V, BERA R, BALACHANDRAN S, et al. REDUCT: keep it close, keep it cool!: efficient scaling of dnn inference on multi-core CPUs with near-cache compute[C]//The 48th Annual International Symposium on Computer Architecture. Valencia: IEEE, 2021:167-180.

[23] SCHUIKI F, ZARUBA F, HOEFLER T, et al. Stream semantic registers: a lightweight RISC-V ISA extension achieving full compute utilization in single-issue cores [J]. IEEE Transactions on Computers,2020,70(2):212-

227.
[24] CHEN T, DU Z, SUN N, et al. Diannao: a small-footprint high-throughput accelerator for ubiquitous machine-learning[J]. ACM SIGARCH Computer Architecture News, 2014, 42(1): 269-284.
[25] HAN S, KANG J, MAO H, et al. ESE: efficient speech recognition engine with sparse LSTM on FPGA[C]//Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. Monterey: Association for Computing Machinery, 2017: 75-84.
[26] ABALI B, BLANER B, REILLY J, et al. Data compression accelerator on IBM POWER9 and z15 processors: industrial product[C]//2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture. Valencia: IEEE, 2020: 1-14.
[27] CHEN L, TARANGO J, MITRA T, et al. A just-in-time customizable processor[C]//2013 IEEE/ACM International Conference on Computer-Aided Design. San Jose: IEEE, 2013:524-531.
[28] BANSAL R, KARMAKAR A. Closely-coupled lifting hardware for efficient DWT computation in an SoC[J]. Journal of Signal Processing Systems, 2020, 92(2): 225-237.
[29] GONZALEZ R E. A software-configurable processor architecture[J]. IEEE Micro, 2006, 26(5): 42-51.
[30] CLARK N, KUDLUR M, PARK H, et al. Application-specific processing on a general-purpose core via transparent instruction set customization[C]//The 37th International Symposium on Microarchitecture. Portland: IEEE, 2004: 30-40.
[31] SCHUIKI F, ZARUBA F, HOEFLER T, et al. Stream semantic registers: a lightweight RISC-V ISA extension achieving full compute utilization in single-issue cores[J]. IEEE Transactions on Computers, 2020, 70(2): 212-227.
[32] AMID A, BIANCOLIN D, GONZALEZ A, et al. Chipyard: integrated design, simulation, and implementation framework for custom SoCs[J]. IEEE Micro, 2020, 40(4):10-21.
[33] MERICAS A, PELEG N, PESANTEZ L, et al. IBM POWER8 performance features and evaluation[J]. IBM Journal of Research and Development, 2015, 59(1):1-6.

Tightly coupled heterogeneous thread processor

LI Wenqing, QI Han, XIAO Ziyuan, ZHU Weipu, WANG Jian
(State Key Laboratory of Processors, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)
(School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing 100049)

Abstract

Heterogeneous computing provides a new idea and direction for the system to achieve a higher performance-power ratio. However, a large amount of control signal and data interaction is always an important bottleneck for the system performance in the process of central processing unit (CPU) and accelerator cooperatively executing tasks in heterogeneous systems. To address this problem, a tightly coupled heterogeneous thread processor architecture is proposed, which includes a hardware CPU thread and a hardware accelerator thread. Both of them use pipeline tightly coupled hardware thread communication interface and shared memory to reduce the communication cost and greatly improve the system performance. To verify the advantages of this architecture, a hardware inter-thread communication interface is designed based on the open source BOOM core, a tightly coupled heterogeneous thread processor is implemented with an advanced encryption standard (AES) encryption and decryption accelerator, and evaluation is performed on field programmable gate array (FPGA). The results show that in encryption tasks, the processor throughput is about 4.7 times higher than that of Intel Comet Lake using AES new instructions (AES-NI) in encryption tasks and 4000 times higher than that of the BOOM platform using only general-purpose instructions. The experiment further verifies that more performance gains can be achieved with fine-grained parallelism obtained by fast communication between the CPU and accelerator. This leads to the conclusion that the architecture can agilely integrate the accelerator around the CPU, effectively reduce the communication time and achieve fine-grained parallelism of CPU threads and accelerator threads, effectively exploit the advantages of heterogeneous computing, and obtain considerable performance gains.

Key words: heterogeneous computing, heterogeneous interface, tight coupling, communication, fine-grained parallelism