doi:10.11887/j.cn.202002002 http://journal.nudt.edu.cn

面向图计算应用的处理器访存通路优化设计与实现*

张 旭[1,2],常轶松[1,2,3],张 科[1,2,3],陈明宇[1,2,3]

(1. 中国科学院计算技术研究所,北京 100190;2. 中国科学院大学,北京 100049;3. 鹏城实验室,广东 深圳 518000)

摘 要:针对图计算应用的访存特点,提出并实现一种支持高并发、乱序和异步访存的高并发访存模块(High Concurrency and high Performance Fetcher, HCPF)。通过软-硬件协同的设计方法,HCPF可同时处理192条共8种类型的内存访问请求,且访存粒度可由用户定义,满足图计算应用对海量低延迟细粒度数据访问的需求。同时,HCPF扩展了基于内存语义的跨计算节点定制互连技术,支持远程内存的细粒度直接访问,为后续实现分布式图计算框架提供技术基础。结合上述两个核心研究内容,基于流水线RISC-V处理器核,设计并实现了可支持HCPF的RISC-V片上系统(System-on-Chip,SoC)架构,搭建基于FPGA的原型验证平台,并使用自研测试程序对HCPF进行初步性能评测。实验结果表明,HCPF相比原有访存通路,最高可将基于数组和随机地址的两种随机内存访问性能分别提升至3.5倍和2.7倍。远程内存直接访问4 Byte数据的延时仅为1.63 μs。

关键词:内存级并行;访存通路;图计算应用

中图分类号:TN95 **文献标志码**:A **文章编号**:1001-2486(2020)02-013-10

Design and implementation of a novel off-chip memory access path for graph computing

ZHANG Xu[1,2], *CHANG Yisong*[1,2,3], *ZHANG Ke*[1,2,3], *CHEN Mingyu*[1,2,3]

(1. Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China;
2. University of Chinese Academy of Sciences, Beijing 100049, China; 3. Peng Cheng Laboratory, Shenzhen 518000, China)

Abstract: A novel asynchronous memory access path, which supports highly concurrent and out-of-order off-chip memory requests was proposed. In order to satisfy the requirements of graph applications, a software-defined interface in our proposed memory access path to handle hundreds of kinds of off-chip memory requests with arbitrary granularity via hardware-software co-design methodology was implemented. A custom memory semantic interconnect was designed for fine-grained remote memory access among various computing nodes leveraged in future distributed graph processing scenarios. Last but not least, we integrate our proposed novel memory access path into a RISC-V instruction set architecture-based SoC(system-on-chip) architecture and implement an FPGA prototype. Based on our custom random access microbenchmarks, preliminary evaluation results show that performance of array-based and random address-based off-chip memory access is improved by 3.5x and 2.7x respectively using our proposed asynchronous memory access path, and accessing 4 bytes data from remote memory only takes 1.63 μs.

Keywords: memory-level parallelism; memory access path; graph computing

互联网的快速发展使得图(graph)成为一种被广泛使用的数据抽象方法,用于表达数据之间关联关系。基于图表示的社会网络、交通网络、文章相似度、疾病暴发路径、学术文章引用关系等数据,以及应用在这些种类繁多的图数据上的最短路径、分类问题、最小割集和连通分支、PageRank及其变种等图算法构成了各种图计算应用[1-3]。随着大数据时代的到来及人工智能应用的兴起,图计算已被广泛应用于机器学习(如图卷积网络、图神经网络等[4])、计算机视觉、模式识别等相关领域。因此,提高图计算应用的处理性能将对未来智能信息产业的发展起到至关重要的作用。

另一方面,基于传统冯·诺伊曼体系结构的通用处理器计算平台是支撑以图计算应用基础软件框架(如Google Pregel[5]、美国加州大学伯克利分校GraphX[6]等)高效运行的重要基础平台。随着图数据规模的不断增长,图应用负载对通用处理器平台的能力提出了更高的要求。如何提高通用处理器平台的图数据处理性能,以应对不断增长的图数据计算规模带来的挑战,是本文关注的

* **收稿日期**:2019-09-15
基金项目:国家重点研发计划资助项目(2017YFB1001602);国家自然科学基金资助项目(61702485);中国科学院青年创新促进会资助项目(2017143)
作者简介:张旭(1996—),男,河南濮阳人,博士研究生,E-mail:zhangxu19s@ict.ac.cn;
张科(通信作者),男,副研究员,博士,E-mail:zhangke@ict.ac.cn

主要研究内容。

图可以表示成一个由顶点集合和边集合组成的二元组,每条边由顶点集合中两个点相连接所构成;每个顶点和每条边都可以被赋予一个权重值。面向真实应用场景的图数据一般规模较大,数据间的关联关系也呈现复杂多变的特点。因此,图计算应用要处理大量结构复杂的顶点和边数据负载。同时,图算法虽然在处理过程中需要对顶点或边进行多轮遍历,但往往对顶点和边只进行计算量较小的简单归约计算,而不是大规模复杂数值运算。因此,图计算应用在通用处理器平台上执行时,具有内存访问密集的特点,并呈现如下访存行为特征:

1)高并发性:遍历顶点或边数据产生的大量数据访问和简单归约计算相结合,会导致应用在短时间内产生大量的访存请求[7]。

2)低局部性:结构复杂多变的图数据导致程序的访存行为具有低局部性的特点[8]。虽然已有大量冗余结构被发现存在于面向真实应用场景的图数据中,但目前图计算应用仍不能通过在运行时准确预测这些冗余结构在图数据中的位置来改善数据访问的局部性[9-10]。

3)细粒度:由于图计算应用中计算操作只使用顶点或边的某一个属性,而这些属性往往只占用几个字节空间,这导致图计算应用的访存行为具有细粒度的特点[7]。

传统的冯·诺伊曼通用处理器体系结构使用统一的片外内存资源来存放指令和数据。在处理如图计算应用为代表的访存密集型应用时,大规模高并发访存请求会使得基于冯·诺伊曼架构的通用处理器在耗费大量的空闲等待周期后,才能获得运算指令需要的数据,导致指令流水线不能被足够数量的计算指令填满,从而造成指令集并行性无法被处理器充分利用。

从上述分析可以看出,数据访存延时已成为影响访存密集型应用处理性能的主要因素。特别是随着处理器与内存之间的性能差距不断增大(即"Memory Wall"问题[11]),高内存访问延时将成为制约图计算性能进一步提升的主要瓶颈。

考虑到未来大数据环境下不断增长的图数据规模,单计算节点有限的内存资源已无法满足图计算应用的需求,多计算节点共享内存资源成为部署大规模图计算应用的主要方式之一。然而,受访问距离和访问协议的影响,远程访问共享内存会进一步增加延迟开销[12-13]。

因此,探索本地和远程内存资源的高效访问机制,对降低访存延迟的影响至关重要。内存访问延迟受物理条件限制,难以进一步降低,充分发掘内存级并行性是提升内存访问性能的一种有效方法[14]。内存级并行是指处理器指令流水线异步提交多条访存请求至内存子系统(包含处理器内部的访存队列、数据缓存、片上互连总线、片外内存访问通路及控制器等部件),并在同一时间继续处理其他无关指令,有效隐藏访存延迟。

针对上述问题,本文以提升通用计算平台的内存级并行性为主要目标,探索内存访问通路的设计优化方法。

1 相关工作

1.1 非阻塞缓存

目前提升内存级并行度的主流方案之一是在访存通路中使用非阻塞高速数据缓存[15-16]。非阻塞缓存基本结构如图 1 所示。当非阻塞缓存命中时(步骤[h1]至[h3]),非阻塞缓存向处理器返回保存在内部的数据;当非阻塞缓存未命中时(步骤[m1]至[m6]),非阻塞缓存通过将未命中的访存请求保存在 MSHR 数组中,允许处理器继续执行指令,直至当前指令与之前的访存指令之间存在数据依赖。采用非阻塞缓存技术,可使单发射流水线处理器和乱序处理器均得到 15% 以上的性能提升[17-18]。

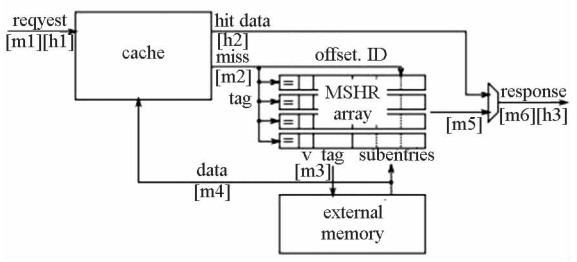

图 1 非阻塞高速缓存结构[19]

Fig. 1 Structure of an non-blocking cache[19]

然而,已有研究工作表明,MSHR 的数量存在上限(一般为十余个)。由于非阻塞缓存使用硬件逻辑完成所有的处理器异步访存操作,且位于访存通路的关键路径上,因此,继续增加 MSHR 的数量并不能进一步提高访存性能,反而会导致处理器面积增大和时钟频率下降等问题,加大了通用处理器的设计难度[17-18]。

针对非阻塞缓存面临的问题,Asiatici 等提出一种新型的表结构 cuckoo hash table[19],用于保存 MSHR,使非阻塞缓存可以快速查找访存请求的信息,提升非阻塞缓存的可扩展性。然而,cuckoo hash table 并未有效解决目前计算系统处

理图计算应用时存在的带宽浪费的问题。

1.2　预取单元

与优化非阻塞缓存微体系结构的方法不同,以下工作尝试根据图数据的特点设计专用预取单元。针对特殊的图表示形式,有限深度优先调度算法 BDFS 和相应的 HATS 预取单元尝试提升深度优先搜索中的内存级并行性[8]。HATS 通过查询图顶点的状态,自动从二级缓存预取未被处理的顶点以及边,并存放在内部的缓冲区中。当处理器发现缓冲区非空时,便从缓冲区中取数据并做运算。Ainsworth 等在预取单元中添加了监控处理器一级缓存访问行为的功能,从而能够根据监控结果自动从二级缓存预取数据并保存在一级缓存中[20]。以此为基础,Ainsworth 等进一步提出基于编译器和操作系统支持的可编程预取的思想。当处理器执行到程序编译时添加的触发指令时,便唤醒预取单元进行数据预取[21]。以上工作均将预取的数据存放在一级缓存中,这极有可能将某些相冲突的热点数据替换出缓存,从而增加了不必要的缓存未命中的次数。此外,它们均针对某种特定的图表示形式,不具备通用性。从二级缓存预取数据的方式,仍然存在着内存级并行性受限于二级缓存内部 MSHR 数量的问题。

1.3　专用加速器

设计图计算专用加速器也是提升图计算应用处理性能的重要手段。例如,图计算加速器 GRAPHR[7] 采用存内计算设计思想,使用新型 ReRAM 存储介质内部结构并行处理大量的模拟信号运算,解决了图计算应用面临的内存带宽不足的问题。然而 GRAPHR 仅适用于可表示成稀疏矩阵向量乘的图算法,无法像基于通用处理器的计算平台一样满足多种不同图算法运行的需要。

图数据应用的低局部性特点不仅会影响缓存的命中率,也会降低存储设备的性能。针对这一问题,基于闪存存储设备的图计算加速器 GraphBoost[22] 将海量的图数据保存在外部闪存中,只用固定的内存资源保存部分图数据,避免了 GraphBoost 的性能受图数据规模的影响。当处理器需要遍历闪存中的数据时,GraphBoost 记录访存的地址,然后顺序的访问这些数据并执行递归操作。然而,GraphBoost 在较大规模的数据上才能取得较好的性能提升。

1.4　设计原则

结合以上工作的优缺点以及图计算应用的访存特点,HCPF 的设计需要考虑以下 5 个方面:

1)图计算应用的高并发访存需要大量的具有 MSHR 功能的部件,才能减少处理器流水线被访存指令暂停的概率,进而提升内存级并行。因此,HCPF 中具有 MSHR 功能的部件应有好的扩展性,即其数量不严重受限于硬件资源。

2)非阻塞缓存的访存粒度固定且远大于图计算应用的需求,又因为访存的低局部性特点,导致缓存行不会被再次命中。因此,内存访问的粒度需要由软件定义,以减少带宽的浪费。

3)非阻塞缓存使用硬件逻辑完成异步访存功能,导致其可扩展性不高。因此,为了降低硬件逻辑的复杂度同时提升 HCPF 的可扩展性以及实现软件定义访存粒度的功能,本文尝试通过软硬件结合的方式完成异步访存的功能。

4)由于单计算节点内部内存容量的限制,多计算节点间共享使用内存资源成为不可避免的趋势。远程内存直接访问可以共享更多的内存资源,实现动态负载平衡。此外,图计算应用中访存粒度小的特性,导致目前的远程访问协议(如 TCP/IP)会产生大量的通信开销以及带宽浪费。因此,HCPF 需要具备高效的远程内存直接访问的能力。

5)由于访问不同计算节点的内存资源产生的延迟依赖于距离、带宽和传输介质等多个因素,导致访存延迟的波动较大。因此,HCPF 需要通过乱序访存的方式,保证访存的总时间最短。

2　访存通路软硬件设计

2.1　访存通路硬件框架

访存通路分为高并发访存模块(HCPF)和远程内存直接访问模块(DoCE)两部分,硬件框架如图 2 所示。通用处理器中的指令窗口允许暂存未完成的访存指令或存在数据依赖的运算指令,进而使处理器继续执行其余指令,但是,由于图计算应用中存在海量且高延迟的访存请求,远超过目前指令窗口的容量,为避免流水线暂停,处理器需要另一种方式暂存这些数量庞大的访存指令。一种方式是用写寄存器替代读指令,既可避免指令窗口缓存写指令,允许处理器继续执行,又通过将访存请求交由 HCPF 处理,允许在 HCPF 内部实现特定的与处理器结构无关的优化。为支持处理器异步地将访存请求写入 HCPF 的内部寄存器,HCPF 分为前端和后端两个部分,前端包含前端控制寄存器、读数据缓冲区和写数据缓冲区;后端包含读请求表、写请求表和后端控制器。HCPF 负责解码处理器的访存指令,并访问本地内存,或将访存请求发往 DoCE。DoCE 负责访问远程内存资源。

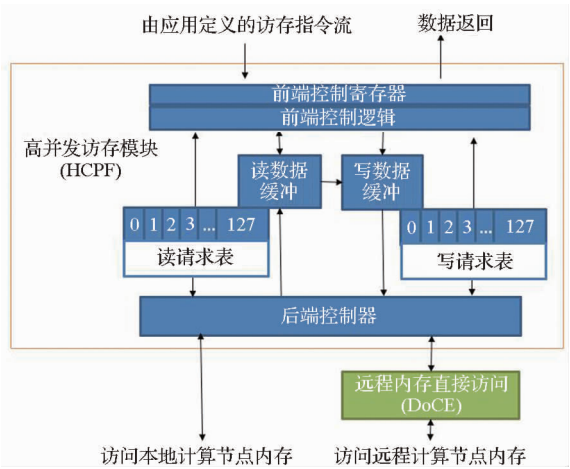

图 2　访存通路的硬件框架设计

Fig. 2　Structure of the memory access path

2.2　HCPF 前端

HCPF 前端负责解码处理器请求、答复处理器请求、读写内部缓冲区以及更新请求表。

控制单元中有一组内存地址映射的寄存器，这组地址对应的数据不会被保存在缓存中。处理器既可向某个地址写入访存指令，亦可从这组地址获得 HCPF 状态信息。

由于图数据中顶点和边的数据规模固定，且一个顶点或边的结构体内部存在良好的局部性，因此，HCPF 支持处理器以顶点或边的粒度访问内存。此外，图的链接信息以数组下标或者链表的形式保存，图算法一般通过链接信息遍历图，而不是随意的选择顶点，本文针对这一特点，设计收集数据指令，允许处理器以一条指令访问多个顶点。由于图算法中的归约运算一般不会涉及复杂的访存指令，因此，HCPF 目前支持 8 种访存指令，包含读数据指令、写数据指令、收集数据指令、缓冲区指令、操作完成指令和写回指令等。

控制单元根据访存指令产生一系列的控制信号和访存请求，例如将读数据缓冲区的数据返回处理器、写入写数据缓冲区、将读数据缓冲区的数据搬运到写数据缓冲区、更新写请求表和读请求表等。

2.3　HCPF 后端

HCPF 后端负责读写内部缓冲区、发送访存请求以及接收内存返回的数据。

读、写请求表中除保存访存请求外，还包含一系列状态寄存器，保存读取地址未发送表项地址、写入地址未发送的表项地址和写入数据未发送的表项地址等。此外，针对图计算应用中 3 种数据需求（边、当前顶点、相邻顶点），HCPF 中有相应的 3 种读请求表，通过这样的设计，保证了某一类访问频繁的数据不会将其他种类的数据替换出请求表。由于顶点或边的结构体内部存在局部性，那么处理器可以通过数据复用以减少执行时间，因此，HCPF 允许处理器决定某次访存指令位于请求表的位置以及表项的生存期，并通过请求表访问缓冲区中的数据。

由于目前商用的总线协议（例如 AXI）采用读写通道分离的方式，支持显著传输模式，以降低传输延迟，因此，后端控制器内部有 4 个通道，分别是读地址通道、读数据通道、写地址通道和写数据通道，可乱序异步地向内存控制器发送地址和接发数据。

若读请求表非空时，后端控制器从读请求表中获得读取地址未发送表项，并将相关信息发送给内存控制器，如图 3 所示。

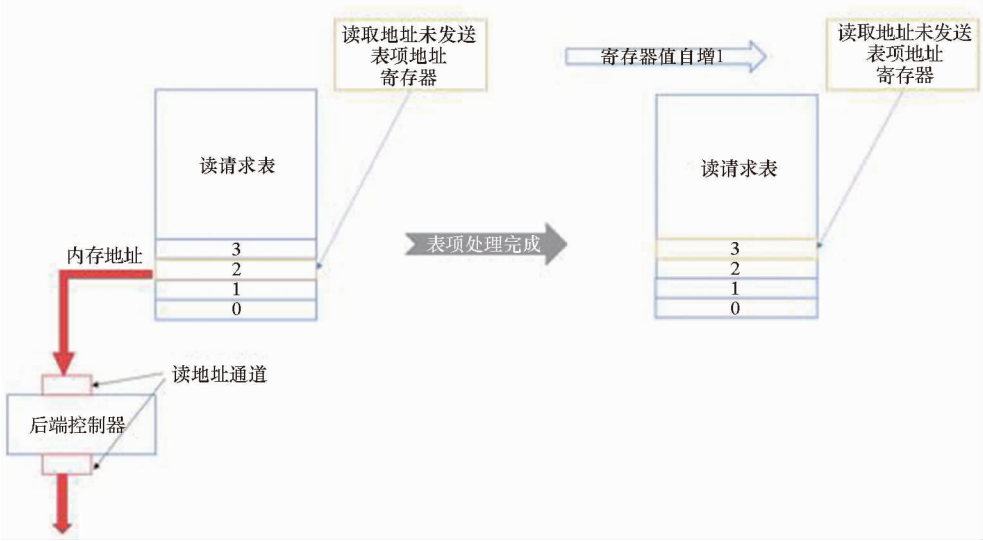

图 3　读请求表非空时处理流程

Fig. 3　Flow graph when the reading-request table is not empty

后端控制器收到内存答复时,根据返回数据的标记,计算出对应的读请求表表项地址,然后将数据写入对应的读数据缓冲区地址,如图 4 所示。

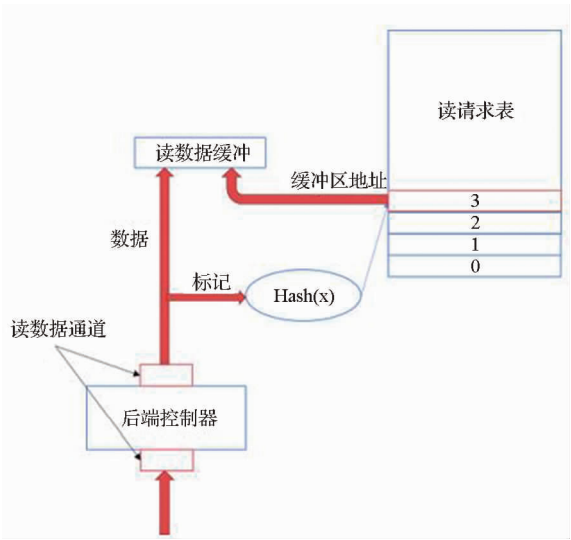

图 4　内存答复处理流程

Fig. 4　Flow graph when back-end controller received memory replies

若写请求表非空时,后端控制器从写请求表中获得写入地址未发送表项和写入数据未发送表项以及从写数据缓冲区获得需写入的数据,并将相关信息发送给内存控制器,如图 5 所示。

2.4　节点间内存语义互连

图计算应用中,处理器会产生大量细粒度的访存请求,若在节点间采用基于 TCP/IP 协议的连接方式,则会由于 TCP/IP 包的控制信息长度大于数据段长度,导致带宽的浪费。此外,TCP/IP 协议栈的处理复杂,时间延迟长,进一步增加了访存延迟。

基于共享内存的内存语义互连[23]通过将片上总线直接扩展(DEOI)[24]协议部署在以太网上,实现不同计算节点之间通过以太网链路层协议的远程互连。内存语义互连相较于传统的 TCP/IP 协议,传输数据的延迟受数据规模的影响小。片上总线直接扩展协议[24]支持用户使用片上互连协议访问相邻计算节点的资源,使得远程资源如同本地资源。DEOI 仅允许单个计算节点访问远程内存资源,并令所有计算节点共享内存资源划分信息。

HCPF 选择基于共享内存的内存语义互连作为节点间互连方式。如图 2 所示,后端控制单元将涉及远程计算节点的访存请求发送给内存语义互连模块(Direct extension of on-chip interconnects over Converged Ethernet, DoCE, 绿色部分)。DoCE 具有两部分功能,一是使用 MAC 协议封装访存请求,然后通过以太网发往远程计算节点的 DoCE;二是对远程计算节点发来的访存请求进行解码,进而访问本地内存。

2.5　基于 HCPF 的性能测试程序设计

如引言部分所述,图计算应用的数据局部性难以发掘,因此,本文假设图数据应用的访存行为是随机的。多数图计算应用会选择以顶点为中心的计算模型,例如 Pregel[5]和 GraphLab[25]。这些模型的顶点数据保存在数组结构中,边数据保存在链表结构或邻接矩阵中。因此,这种图计算应用中的访存行为近似于对数组元素的随机访问。为了充分模拟上述访存行为,本文设计并实现一种基于数组的随机访问测试程序,见算法 1。

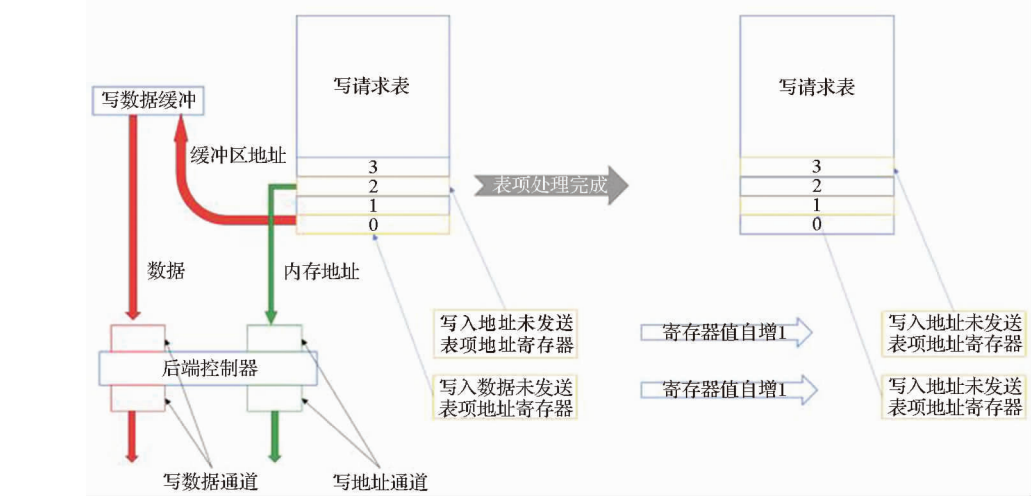

图 5　写请求表非空时处理流程

Fig. 5　Flow graph when the writing-request table is not empty

算法 1 基于数组的随机访问测试程序

Alg. 1 Array based random access testbench

```
Input:count
Input:array and the length of the array
While count > 0 do
    index←Random( ) % length  //随机产生元素下标
    sum←sum + array[ index]
    array[ index] ←sum
    count←count - 1
end
Output:sum
```

此外,针对使用邻接表保存图数据的图计算应用,本文设计并实现一种基于随机地址的随机访问测试程序,见算法 2。

算法 2 基于随机地址的随机访问测试程序

Alg. 2 Random address based random access testbench

```
Input:count
Input:base and size of the address space
While count > 0 do
    Address←base + Random( ) % size  //随机产生地址空间内的地址
    sum←sum + * Address
    * Address←sum
    count←count - 1
end
Output:sum
```

为了使上述两个算法用于 HCPF,本文针对 HCPF 设计了一组软件接口以及地址的宏定义,以隐藏硬件的设计细节,并对测试程序进行修改。

3 基于 HCPF 的片上系统框架

3.1 片上系统框架

本论文的单个计算节点的总体框架如图 6 所示。片内主要包含处理器、系统总线、内存总线、HCPF 和 DoCE;片外主要包含内存和数据传递模块。本文的设计为图中蓝色部分。

处理器通过内存总线或 HCPF 访问本地内存;亦可进一步通过 DoCE 访问远程内存;远程计算节点可通过本地 DoCE 模块访问本地的内存。

3.2 通用处理器微结构的选择

HCPF 需要通用处理器核产生高并发的访存请求,才能充分发挥其性能优势。对于同一个程序,访存指令的数量固定,但是由于乱序处理器具有较高的指令并行度,一定时间内提交给访存通路的访存请求数可能更多。然而,程序的指令顺序是可以修改的,我们可以通过让程序先执行全部访存指令,使得顺序处理器和乱序处理器在一定时间内提交相同数量的访存请求。上述分析也适用于单发射和多发射处理器的对比。因此,不同微结构的处理器并发地产生多条访存请求的能力是一致的[26]。虽然乱序处理器内部通过指令窗口缓存未完成的指令,一定程度上提升处理器应对高延迟访存的能力,然而,相关论文表明目前商用处理器的指令窗口规模仍不足以处理海量的高延迟访存指令[27],因此 HCPF 产生的性能效益适用于多种通用处理器,基于 HCPF 的片上系统架构不依赖于处理器核的微结构。

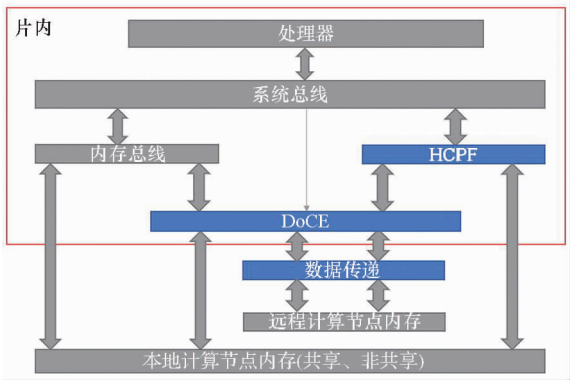

图 6 基于 HCPF 的片上系统架构

Fig. 6 Architecture of HCPF based SoC

HCPF 可以通过与缓存协同工作,各自发挥自身优势,实现性能提升“1 + 1 > 2”的效果。本文仅考虑图计算应用场景下二者的协同工作方式。图计算应用中,软件可以将涉及图数据的访存交由 HCPF 处理,剩余少部分访存交由非阻塞缓存处理,此时 MSHR 数量对性能的影响可以忽略不计。因此,用户可以根据硬件资源以及应用场景定制 MSHR 的数量。本文将在第 4 节对配置不同数量 MSHR 的片上系统进行图计算应用场景下的性能对比。

3.3 内存模型

由于本文只将此片上系统框架应用于简单的测试程序,故只考虑 HCPF 内部的内存模型,不考虑和原有访存通路之间的一致性问题。根据第 2.3 节所述,HCPF 将读写操作分通道并行执行,故不保证读写操作的一致性,但是分别保证写与写、读与读之间的一致性。计算节点之间可以通过冲刷操作(Flush),将本地修改的数据更新到相应的远程节点。

3.4　总线接口设计

如图 7 所示,片上系统框架中需要 5 个内存映射区域,分别是本地计算节点非共享内存、本地计算节点共享内存、HCPF、DoCE,以及远程计算节点内存。为了适应图计算应用的高并发访存以及最大化 HCPF 的性能,所有内存区域的总线接口都需要支持并发访问。由于 HCPF 实现了乱序访存机制,本地以及远程内存区域的总线接口只需保证相同标记的数据有序返回。

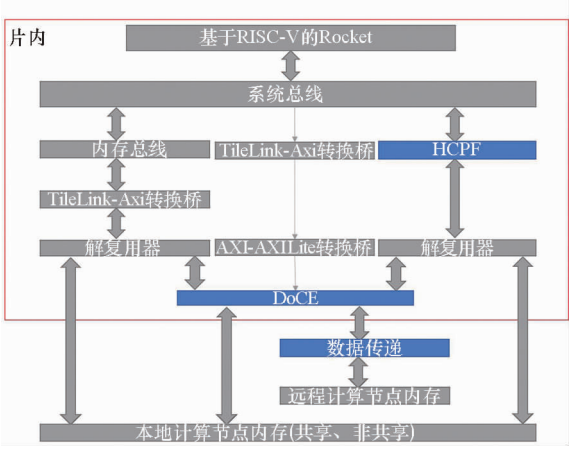

图 7　本文实验条件下的片上系统

Fig. 7　HCPF based SoC in our platform

4　实验结果与评估

4.1　基于 RISC-V 开放指令集的片上系统配置

如 3.2 节所述,本文选择基于 RISC-V 指令集的 Rocket Chip[28] 片上系统生成器,采用支持 RV32G 和 RV64G 指令集的 64 位单发射处理器核 Rocket 作为片上系统的通用处理器核。本文使用 Chisel 语言[29](Constructing hardware in a scala embedded language)描述 HCPF 的硬件逻辑,经解释编译后生成可综合的 Verilog 文件。

片上系统的主要配置见表 1。

表 1　片上系统的配置

Tab. 1　Configuration of the SoC

参数名称	参数值
缓存字节块大小	64 Byte
缓存大小	16 KByte
MSHR 数量	0
HCPF 的缓冲区大小	4 KByte
HCPF 的请求表表项个数	64
节点内数据传输位宽	64 bit
节点间数据传输位宽	64 bit

片内采用 TileLink 作为互联协议。此外,除了自研访存通路外,片上系统还需要其他通用 IP 核,由于这些 IP 核多采用 AXI 接口,本文选择 AXI 协议作为片外互连方式。处理器通过 AXI-Lite 协议配置 DoCE。为了缩短计算节点之间的传输延迟,DoCE 使用 AXI-Stream 协议直接访问远程内存。

在本文的实验平台下,基于 HCPF 的片上系统如图 7 所示,蓝色部分为本文的设计。TileLink-AXI 协议转换桥与 AXI-AXI Lite 转换桥负责不同连接协议之间的转换。

4.2　片上系统的 FPGA 实现

本论文选择在 Zynq - 7000 系列中的 ZC706[30] 板卡上部署基于 HCPF 的片上系统,搭建完成的双计算节点系统如图 8 所示。“1”为电源线;“2”为 LED 灯,灯亮指示节点间已建立连接;“3”为连接节点的万兆光纤;“4”为串口线,负责接收主机的控制信号以及传输调试信息;“5”为以太网线,负责在板卡与主机之间传输文件。

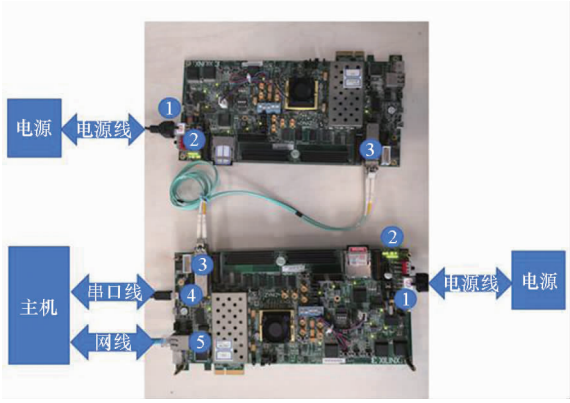

图 8　部署 HCPF 的双节点分布式系统

Fig. 8　The HCPF based distributed system with two processing nodes

4.3　基础软件部署

本文在片上系统上部署基于 RISC-V 的轻量级程序运行环境 Proxy Kernel,并在 ZC706 板卡的 ARM 处理器核上运行调用 Fesvr 库的程序Zynq - fesvr,以实现 Rocket 与 ARM 处理器核之间的交互。

4.4　基于随机访问测试程序的性能评估

本文在系统上运行 2.5 节设计的测试程序,比较不同配置的系统完成相同程序需要的时钟周期数。由于顶点的值或边的权重一般为 32 位或者 64 位浮点数,所以令每次随机访问的粒度为 8 字节。设测试程序循环的次数为 2000 次,访问本地内存的性能结果如图 9 所示。

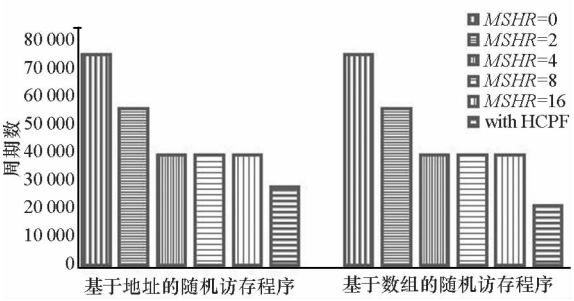

图 9 不同配置的片上系统在两种测试程序上消耗的周期数

Fig. 9 CPU cycles of different SoC configurations running on two testbenches

根据图 9 所示,部署 HCPF 的片上系统在基于随机地址的随机访存程序中性能提升到 1.4 ~ 2.7 倍,在基于数组的随机访存程序中性能提升到 1.8 ~ 3.5 倍。此外,当系统中 MSHR 的数量由 4 增为 16 后,系统的性能并没有明显的改变。

由于系统涉及处理器、总线、内存等多个器件,定量分析性能提升的原因较复杂,以定性分析的角度,系统性能的提升在于处理器与 HCPF 之间良好的配合,降低了处理器访存的延迟同时减少流水线暂停的次数。处理器以写寄存器代替读指令,提前将需要的数据以写寄存器形式通知 HCPF,极大地减少流水线暂停的次数。HCPF 根据写入的指令,将节点或者边读取至缓冲区中,使得处理器在后续的处理中可以及时地读取数据,降低了访存延迟。

为探究 HCPF 的可扩展性,本文比较不同 MSHR 数量的缓存和不同请求表表项个数的 HCPF 分别占用的板卡资源,如图 10 和图 11 所示,本文仅展示有明显改变的 LUT 和 LUT RAM 资源。

根据图 10 和图 11,将系统中 MSHR 的数量从 0 逐渐增为 16,占用的查找表(LUT)资源增加约 4%,占用的 LUT RAM 资源增加约 3%;相应地,将 HCPF 中请求表的表项个数从 96 增加到 192,占用的 LUT 资源与 LUT RAM 资源均增加接近 1%。因此,HCPF 相较传统访存通路在访存并发度上更具备可扩展性。

通过 DoCE 访问远程内存资源的性能与通过内存总线访问本地内存的性能对比如表 2 所示。由于实验平台采用 64 位处理器,故无法使用一条访存指令通过内存总线访问 32 字节的数据,因此本文未测量处理器以 32 字节的粒度访问本地内存的往返时延。从表中可以看出,通过 DoCE 传输

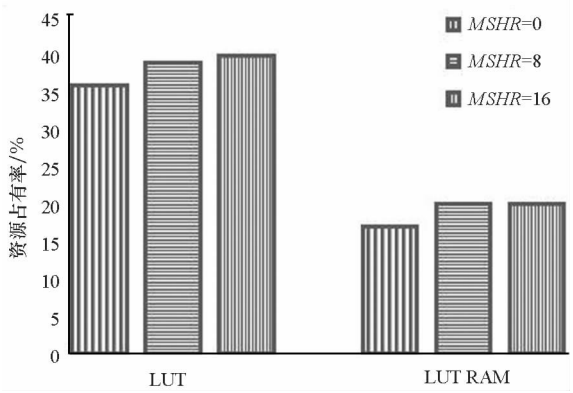

图 10 配置不同数量 MSHR 的片上系统在两种板卡资源上的资源占用率

Fig. 10 Utilization rate of LUT and LUT RAM of different MSHR configurations

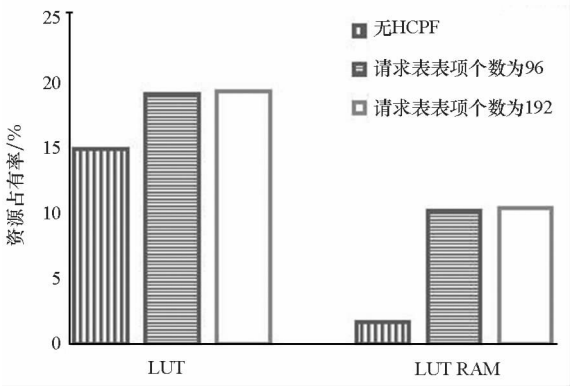

图 11 配置不同规格 HCPF 的片上系统在两种板卡资源上的资源占用率

Fig. 11 Utilization rate of LUT and LUT RAM of different HCPF configurations

数据的延迟受数据规模的影响小,比访问本地内存的时延高 1.23 μs,可有效避免通信产生的时间开销。

表 2 访问本地或远程内存的传输延迟与数据规模的关系

Tab. 2 Transmission delay of the local and remote memory access on different data size

	往返时延/μs	
	32 bit 数据	32 Byte 数据
远程内存	1.63	2.00
本地内存	0.40	不适用

为探究当前设计存在的性能瓶颈,本文测试了连续读写 HCPF 的寄存器所需周期数,如表 3 和表 4 所示。

可以看出,当程序产生多次写操作时,处理器读写 HCPF 寄存器大约需要 6 个周期。

表 3　周期数与连续写 HCPF 内部寄存器次数的关系

Tab. 3　Relationship between the CPU cycles and the number of consecutive writes of HCPF internal memory mapped registers

写次数	1	2	3	4	5	6
周期数	3	9	16	21	27	33

表 4　周期数与连续读 HCPF 内部寄存器次数的关系

Tab. 4　Relationship between the CPU cycles and the number of consecutive reads of HCPF internal memory mapped registers

读次数	1	2	3	4	5	6
周期数	3	9	16	23	31	37

所以,本文认为当前的设计存在两处性能瓶颈。首先是处理器访问 HCPF 中寄存器的并发度不足,导致处理器平均每 6 个周期才能完成一个写操作,然而 HCPF 处理每一条请求只需要 1 个周期。其次是,处理器向 HCPF 写入访存请求的操作需要拆解成多条指令。例如,对于读数据请求,处理器为填充不同的字段至少需要 3 条指令,若如上文所述,写 HCPF 的寄存器需要 6 个周期,则总共要 9 个周期(4 条指令)才能发送一条读数据请求。

5　结论

围绕图计算应用中大量高并发、低局部性和细粒度的内存访问请求所带来的高访存延时问题,本文探索了内存访问通路的设计和优化方法,通过与缓存所在访存通路协同工作,有效提升图计算应用性能的效果,并完成如下工作:

1)通过在 HCPF 中实现异步访问机制、百余条访存请求缓冲及乱序处理逻辑,满足高并发访存需要;同时通过将返回数据保存在 HCPF 内部缓冲区的方法,有效避免低局部性访问请求污染数据缓存。

2)通过软 - 硬件协同设计的方法,使 HCPF 支持读/写操作、写回、收集等类型的访存请求;同时允许用户定义请求访问粒度,适应图计算应用细粒度内存访问的需要,有效避免内存访问带宽浪费。

3)设计基于内存语义的跨计算节点定制互连技术,支持远程内存的细粒度直接访问,为后续实现分布式图计算框架提供了技术基础。

4)设计实现可支持 HCPF 的 RISC-V 指令集处理器片上系统架构,搭建了基于 FPGA 的原型验证平台,并使用自研测试程序进行性能评测。实验结果表明,本文提出的 HCPF 优于目前的访存通路。

参考文献(References)

[1] Satish N, Kim C, Chhugani J, et al. Large-scale energy-efficient graph traversal: a path to efficient data-intensive supercomputing [C]//Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. IEEE Computer Society Press, 2012: 14.

[2] Hong S, Chafi H, Sedlar E, et al. Green-Marl: a DSL for easy and efficient graph analysis [J]. ACM SIGPLAN Notices, 2012, 47(4): 349 - 362.

[3] Murphy R C, Wheeler K B, Barrett B W, et al. Introducing the graph 500[J]. Cray Users Group (CUG), 2010, 19: 45 - 74.

[4] Zhang Z W, Cui P, Zhu W W. Deep learning on graphs: a survey[R]. arXiv Preprint arXiv:1812.04202, 2018.

[5] Malewicz G, Austern M H, Bik A J C, et al. Pregel: a system for large-scale graph processing[C]//Proceedings of the ACM SIGMOD International Conference on Management of data, 2010: 135 - 146.

[6] Gonzalez J E, Xin R S, Dave A, et al. Graphx: graph processing in a distributed dataflow framework [C]// Proceedings of 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI '14), 2014: 599 - 613.

[7] Song L H, Zhuo Y W, Qian X H, et al. GraphR: accelerating graph processing using ReRAM [C]// Proceedings of IEEE International Symposium on High Performance Computer Architecture (HPCA), 2018: 531 - 543.

[8] Mukkara A, Beckmann N, Abeydeera M, et al. Exploiting locality in graph analytics through hardware-accelerated traversal scheduling [C]//Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2018: 1 - 14.

[9] Leskovec J, Lang K J, Dasgupta A, et al. Statistical properties of community structure in large social and information networks [C]//Proceedings of the 17th International Conference on World Wide Web. ACM, 2008: 695 - 704.

[10] Beamer S, Asanovic K, Patterson D. Locality exists in graph processing: workload characterization on an ivy bridge server[C]// Proceedings of International Symposium on Workload Characterization. IEEE, 2015: 56 - 65.

[11] Wulf W A, McKee S A. Hitting the memory wall: implications of the obvious[J]. ACM SIGARCH Computer Architecture News, 1995, 23(1): 20 - 24.

[12] Jia C F, Liu J N, Jin X, et al. Improving the performance of distributed tensorflow with RDMA[J]. International Journal of Parallel Programming, 2018, 46(4): 674 - 685.

[13] Dang H V, Dathathri R, Gill G, et al. A lightweight communication runtime for distributed graph analytics[C]// Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2018: 980 - 989.

[14] Farkas K I, Jouppi N P. Complexity/performance tradeoffs with non-blocking loads[R]. Western Research Laboratory,

California, USA, 1994.

[15] Chen T F, Baer J L. Reducing memory latency via non-blocking and prefetching caches [C]// ASPLOS V: Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, 1992: 51 - 61

[16] Kroft D. Lockup-free instruction fetch/prefetch cache organization[C]//Proceedings of the 8th Annual Symposium on Computer Architecture. IEEE Computer Society Press, 1981: 81 - 87.

[17] Li S, Chen K, Brockman J B, et al. Performance impacts of non-blocking caches in out-of-order processors [R]. HP Laboratories, 2011.

[18] Tuck J, Ceze L, Torrellas J. Scalable cache miss handling for high memory-level parallelism[C]//Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture. IEEE Computer Society, 2006: 409 - 422.

[19] Asiatici M, Ienne P. Stop crying over your cache miss rate: handling efficiently thousands of outstanding misses in FPGAs[C]//Proceedings of the ACM/SIGDA International Symposium on FPGA. ACM, 2019: 310 - 319.

[20] Ainsworth S, Jones T M. Graph prefetching using data structure knowledge [C]//Proceedings of the International Conference on Supercomputing. ACM, 2016: 39.

[21] Ainsworth S, Jones T M. An event-triggered programmable prefetcher for irregular workloads[C]//Proceedings of ACM SIGPLAN Notices. ACM, 2018, 53(2): 578 - 592.

[22] Jun S W, Wright A, Zhang S Z, et al. GraFBoost: using accelerated flash storage for external graph analytics[C]// Proceedings of ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), 2018: 411 - 424.

[23] Chang Y S, Zhao R, Yu L, et al. DoCE: direct extension of on-chip interconnects over converged Ethernet for rack-scale memory sharing[C]//Proceedings of the 1st Workshop on Emerging Technologies for Software-Defined and Reconfigurable Hardware-Accelerated Cloud Datacenters. ACM, 2017(4): 1 - 3

[24] Chang Y S, Zhang K, McKee S A, et al. Extending on-chip interconnects for rack-level remote resource access [C]// Proceedings of IEEE 34th International Conference on Computer Design (ICCD), 2016: 56 - 63.

[25] Low Y C, Bickson D, Gonzalez J, et al. Distributed GraphLab: a framework for machine learning and data mining in the cloud [J]. Proceedings of the VLDB Endowment, 2012, 5(8): 716 - 727.

[26] Glew A. MLP yes! ILP no! [R]. ASPLOS Wild and Crazy Idea Session, 1998.

[27] Mutlu O, Stark J, Wilkerson C, et al. Runahead execution: an alternative to very large instruction windows for out-of-order processors [C]//Proceedings of the 9th International Symposium on High-Performance Computer Architecture, IEEE, 2003: 129 - 140.

[28] Asanovic K, Avizienis R, Bachrach J, et al. The rocket chip generator[R]. EECS Department, University of California, Berkeley, 2016.

[29] Bachrach J, Asanovic K, Wawrzynek J. Chisel 3.0 tutorial (Beta)[R]. University of California at Berkeley, Berkeley, United States, 2017.

[30] Xilinx. zynq - 7000 all programmable SoC ZC706 evaluation kit[S/OL]. [2016 - 8 - 31]http://www. xilinx. com.