高技术通讯 2022 年 第 32 卷 第 12 期:1251-1261

doi:10.3772/j.issn.1002-0470.2022.12.005

基于社区结构的图数据预取器设计①

李 策② 章隆兵

(计算机体系结构国家重点实验室(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学院 北京 100190)

摘 要 由于图数据规模庞大且结构不规则,图应用运行时会产生大量高延迟内存访问,大幅度降低了通用处理器的运行效率。本文采用软硬件结合的方式设计了图计算专用预取器,利用图数据访存特点以及社区结构的存储规律,通过对图数据进行混合预取,缩短了图计算访存的延迟,在含有较多社区的图数据集上获得了显著的性能收益。在不同图算法与图数据集上的实验表明,该预取器相对于无预取情况、流式预取器及传统图数据预取器,分别实现了 65% ~176%、6% ~21% 和 4% ~18% 的性能提升。

关键词 图计算;预取器;社区结构;存储规律;及时性

0 引 言

随着大数据时代的到来,图作为一种常见的数据结构,在社交网络、网页搜索、推荐系统等领域得到广泛使用。由于图数据的庞大规模,图计算通常运行在分布式系统上,或借助片外磁盘系统存储数据。许多工作[1-4]针对上述系统进行了优化。但随着内存容量与处理器核数的增加,单机系统也可以高效运行图计算应用,且无须对图数据进行分割,相对于分布式系统,减少了额外的预处理开销。

代表现实世界关系的图数据具有稀疏性,对其进行计算时将产生大量无规则访问,不具备时间局部性与空间局部性,通用处理器的片上缓存结构无法发挥作用。导致图计算运行时存在过多的长延迟内存访问,造成处理器内的访存阻塞,成为图应用在单机系统上运行的主要瓶颈[5]。而对访存数据进行预取是缓解上述问题的有效方法。

传统的数据预取器通过学习应用内访存行为的规律,对未来的访存地址进行预测,并将预测地址的数据从内存或下级缓存中取回。等到真正的访存请求到达时,相应的数据已经存储在对应级别的缓存中,达到了缩短访存延迟的目的。但传统的预取器只能识别地址间存在确定规律的访存流,而图计算中包含大量间接访存行为,即访存地址由其他访存请求取回的数据内容决定。传统的预取器不能识别上述访存模式,无法准确预取图数据[6]。

许多研究设计了面向间接访存以及图数据的专用预取器,一定程度上缓解了图计算的访存压力。而本文利用图数据集内社区中邻顶点的储存规律,设计了相应的图数据预取器,进一步提升了预取图数据的性能收益。

本文的主要贡献有以下 3 个方面。

(1)通过对不同类型图数据集的研究,发现了含有社区结构的图数据集的存储规律。

(2)提出了新的图数据预取器设计方案。利用社区结构中邻顶点的存储规律,对图数据中的顶点数据和边数据进行了混合预取,提升了预取的及时性。同时修正现有图数据预取器的设计缺陷,提升了预取的准确性和覆盖率。

① 中国科学院战略性先导科技专项(C 类)课题(XDC05020100)资助项目。

② 男,1992 年生,博士生;研究方向:计算机系统结构,通用处理器设计,图计算加速;联系人,E-mail:lice@loongson.cn。

(收稿日期:2021-07-19)

(3)在不同类型图算法与图数据集上对上述预取器设计进行验证。结果表明该预取器相对于现有图数据预取器实现了 4 % ~18 % 的性能提升。

1 相关工作

数据预取器是体系结构的热点研究方向之一。但传统的数据预取器如 GHB[7] 和 VLDP[8] 等,通常通过学习地址流中的步长历史进行访存预取,无法识别图计算中的间接访存模式。

除图计算外,间接访存模式也存在于许多大规模应用中,因而许多研究[9-11]提出了面向间接访存的预取器。该类预取器采用软硬件结合的方式,通过对访存取回的数据进行分析,可以识别多层级的间接访存模式。但图计算中的间接访存只有两级,当该类预取器处理图应用时,往往会出现过度预取现象。这浪费了内存带宽,同时也增加了处理器运行功耗。为了减少错误预取,IMP[12] 预取器通过对等步长访存流的识别,将等步长访存流的访问数据与随后的访存地址进行对比。若存在对应关系,则判断当前存在间接访存模式,并发出相应的预取请求。该预取器完全基于硬件实现,无额外的软件开销。同时严格限制间接预取的发出条件,提升了预取的准确率。但它产生了额外的启动开销,降低了预取的覆盖率,且无法识别除上述模式外的间接访存,导致其不能准确预取图计算中的全部访存请求。

为了准确全面地预取图计算中的访存请求,文献[13,14]分别设计了面向图数据的专用预取器。其中文献[13]利用广度优先搜索(breadth-first search, BFS)类图算法具有顶点处理顺序可预测的特点,提升了图数据预取的准确性。通过在算法中添加工作表结构,记录当前图顶点的处理顺序,准确地预测了以压缩稀疏行(compress sparse row,CSR)格式存储的图数据的访存行为。同时增加额外的硬件电路实现了对预取及时性的精确控制,极大地提升了 BFS 类图算法的运行性能。但由于需要在算法中添加额外的数据结构以记录图顶点的处理顺序,产生了对图计算软件框架的修改。面对不同的图计算框架,该方法会造成大量的处理开销。而 DROPLET[14] 预取器使用软件方法在页表缓存中对图数据的虚地址范围进行标记。当标记地址被访问时,触发对图数据的间接预取。该设计将预取器放置在内存控制器附近,当数据从内存到达内存控制器时便触发预取请求,大幅提升了预取的及时性。但该预取器需要在内存控制器处添加页表缓存完成虚实地址转换,增加了额外硬件开销,同时随着处理器核数增加,维护成本将越来越高。且该预取器只对顶点数组进行间接预取,而对边数组采用流式预取,导致部分边数组的访存请求未能被准确识别,降低了预取覆盖率。

2 研究背景

本节主要对图计算的基础知识和图数据的存储特点进行介绍。

2.1 图数据存储格式

CSR 是一种被广泛使用的图数据存储格式,使用该格式存储图数据时空间利用率较高。图 1 为 CSR 存储模式示例,CSR 格式使用 2 个数组存储图数据的空间结构。其中偏移数组存储每个顶点对应的第一条边在边数组中的地址,边数组则按照顶点顺序存储每个顶点作为源顶点时,连接的所有边的目的顶点序号。而图应用运算时产生的数据则存储在顶点数组中。不同的图应用对应的顶点数据内容有所不同,例如在 PageRank 算法中顶点数组存储的是每个顶点对应的网页权重,而单源最短路径算法中存储的则是路径长度。同时在某些需要权重的算法中,每条边的权值也会被存储在边数组中,便于运行时读取。

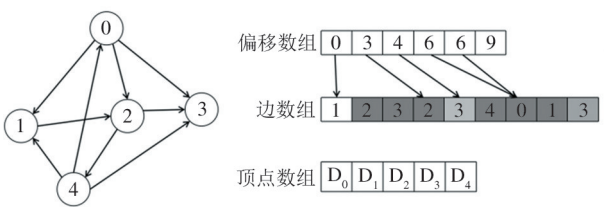

图 1 CSR 存储格式示例

2.2 图计算访存特点

目前许多图计算框架[15]都采用以顶点为中心

(vertex-centric)的计算模式。在该计算模式下,对每个顶点属性值进行计算时,需要读取该顶点相邻顶点的属性值。当使用CSR格式存储图数据时,以PageRank(PR)算法为例,访存过程如下:(1)访问偏移数组读取计算顶点第一条边的偏移地址;(2)在边数组中获取该边的目的顶点编号;(3)读取顶点数组中对应的属性值;(4)对图中所有活跃顶点重复上述操作直至完成计算。

在上述访存过程中,由偏移数组到边数组的访问是第一级间接访存,而由边数组到顶点数组的访问是第二级间接访存。同时每个顶点对应的目的顶点编号顺序存储在边数组中,对其访问时将生成线性访存流。图数据专用预取器需要正确识别上述访存模式,完成对间接访存和线性访存模式的准确预取。

2.3　图数据存储特点

社区结构是许多代表现实世界关系的图数据集的重要特征之一[16-18]。社区是指在图数据中,相对其余顶点联接更为紧密的部分顶点的集合。

本文通过分析不同类型图数据集邻节点的分布规律来研究图数据中社区结构的储存特点。为了表征上述特点,本文提出了2个统计指标:一是储存每个顶点邻节点实际所需缓存行数量A;二是储存每个顶点邻节点所需最小缓存行数量B,B等于邻顶点数量除以每个缓存行所能容纳的最大顶点数量。在图数据集中对上述2个指标分别求和,得出二者的比例C,用于表示图数据中邻节点存储的稀疏度。具体公式如下。

$$C = \sum_{i=0}^{n} A_i / \sum_{i=0}^{n} B_i \tag{1}$$

其中n是图中顶点数量。默认图数据中每个顶点大小为4 bytes,处理器缓存行大小为64 bytes。每个缓存行可以存储16个顶点。当所有邻节点连续分布时,A等于B,C取得最小值1。当每个顶点连接的所有邻节点都属于不同的缓存行时,C取得最大值16。C值越大表明邻节点存储越稀疏,反之邻节点储存越密集。图2是不同图数据集中C值的统计结果,其中数据集berkstan、indochina、arabic的C值皆小于2,说明在上述图数据集中大部分顶点的邻节点存储在邻近位置。数据集google和livej的C值皆高于10,表明其顶点邻节点整体分布较为稀疏。而数据集hollywood的C值介于二者之间,只有部分顶点的邻节点存储较为紧密。

图数据集之所以存在上述邻顶点密集存储的特点,是由于社区中顶点的邻节点大概率属于本社区。在图计算中,连续处理同一社区的顶点可以有效提升访存的时间局部性,许多研究利用上述特点对图计算进行了优化[19-21]。因此储存图数据时,同一社区的顶点被放置在内存中的邻近区域。对于隶属于社区的顶点,其邻顶点大概率分布在相近的存储空间内。对应CSR格式储存的图数据集,相同社区的顶点的属性值连续储存在顶点数组中。

在间接预取过程中,图数据预取器在利用边数组取回的数据对顶点数组进行间接预取时,只能取回边数组当前读取顶点编号对应的顶点属性值。但考虑到图数据集中社区的上述存储特点,预取器可以利用读取的边数组中的邻顶点编号对其余邻顶点储存地址进行猜测,发出对猜测邻顶点的预取,达到提升预取及时性的目的。

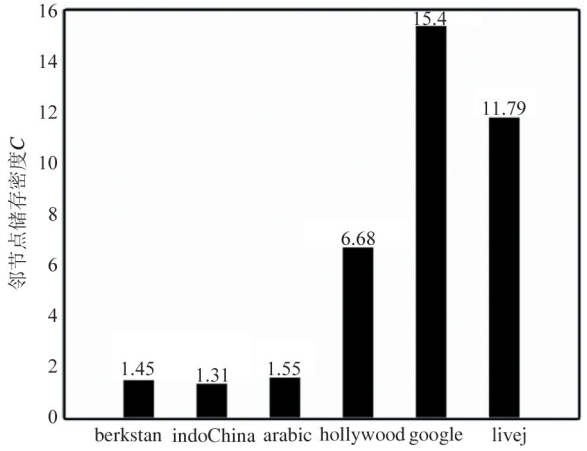

图2　图数据集中C值统计

3　预取器设计

本节将介绍本文所提出的预取器的设计方案和预取过程。下面将本文提出的预取器称为图结构预取器(graph structure prefetcher,GSP)。

3.1　预取器系统级设计

图3为本文预取器系统级设计方案。预取器位于一级数据缓存和二级私有缓存之间,需要监听一

级数据缓存收到的核内访求请求地址,以及由内存返回一级数据缓存的数据,并根据监听到的结果判断是否发射对边数组或顶点数组的预取,如果需要则计算出预取的访存地址并发射。同时将预取器放置于一级数据缓存处,可以节省额外的 TLB(translation lookaside buffer)设计开销。预取请求分为两部分,一是根据内存返回数据发出的对顶点数组和边数组的间接预取请求,该类预取请求直接放入一级数据缓存,这样可以最大程度地减少访问缓存的延迟。同时由于上述间接预取基本是准确的,不会对一级数据缓存造成污染。二是根据边数组返回数据生成的对顶点数组的猜测预取,该类预取有一定的错误概率,因而将其放入二级缓存。

预取器与处理器核间有通路连接,图计算运行时会将偏移数组、边数组和顶点数组的起始地址以及终止地址配置到预取器内的寄存器中。用于判断当前访存地址是否属于图数据,并计算预取的地址。同时预取器也会访问 DTLB(data translation lookaside buffer),预取器中存储的数组地址和内存返回数据生成的地址皆为虚拟地址,所以 GSP 计算出的预取地址也为虚拟地址。需要通过 DTLB 进行虚实地址转换,将对应的物理地址发往缓存。

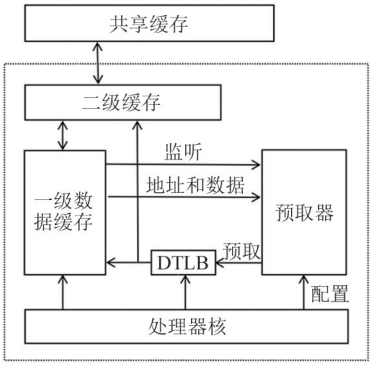

图 3　预取器系统级设计

3.2　预取器构成

如图 4 所示,GSP 预取器由 3 部分组成,分别为图数据地址寄存器、预取缓存队列和预取地址生成器。其中图数据地址寄存器用于存储图数据中 3 类数组的起始地址与终止地址,判断监听到的访存地址是否属于图数据,并参与预取地址的生成。本文通过软硬件结合方式将程序运行时的图数据地址范围配置到上述寄存器中。具体过程为:(1)将预取器内的寄存器设计成用户态可以读写的配置寄存器;(2)在程序代码中添加相应指令将图数据的虚拟地址范围写入该类配置寄存器中。相对于图计算的运行时间,配置图数据虚拟地址所需的软件开销可以忽略不计。

预取地址生成器用于计算预取发出的地址,包括边数组地址生成器和顶点数组地址生成器,最终将生成的预取地址发送到缓存队列中。GSP 中只实现边数组和顶点数组的预取,不对偏移数组进行预取。若实现对偏移数组的间接预取,需要在图应用代码中添加相应的数据结构记录待计算顶点编号,对算法改动较大。同时在规律性迭代类算法如 PageRank 中,对偏移数组的预取可以通过处理器中原有的流式预取器实现。且偏移数组的访存总量占比较小,因而无实现偏移数组间接预取的必要。缓存队列用于缓存生成的预取请求,在处理器空闲时将预取请求发送到 DTLB 中进行虚实地址翻译生成对应的物理地址,然后发往一级及二级缓存。

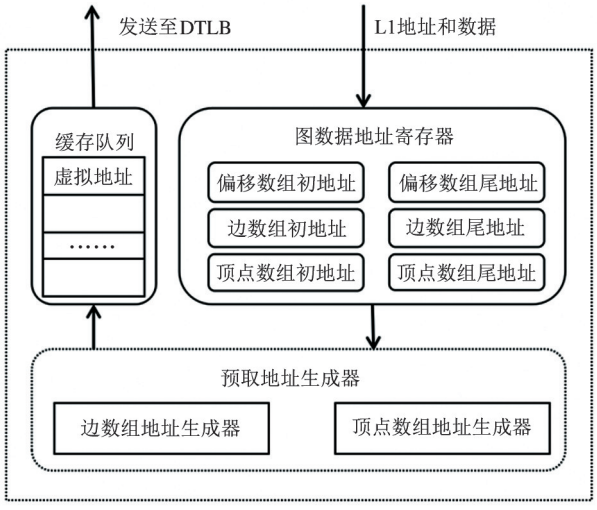

图 4　预取器构成

3.3　顶点数组地址预取

如图 5 所示,在顶点数组预取地址生成器中,顶点数组的预取分成两部分,分别为间接预取和猜测预取。由于图计算中对顶点数组的访问是完全随机的,因而流式预取器无法识别顶点数组的访问模式。如果开启流式预取反而造成缓存污染,所以当访存地址范围属于顶点数组时需要关闭流式预取。

(1) 间接预取。预取器对写回一级数据缓存的地址进行监听,并在图数据地址寄存器中进行比较。当该地址属于边数组时,将被写回的数据写入到顶点数组地址生成器中,并启动对顶点数组的间接预取。边数组返回的数据存储的是顶点编号,假设每个编号生成的预取地址为 PV, 则:

$$PV = vertex\_begin + 4 \times vertex\_id$$

其中 $vertex\_begin$ 代表顶点数组起始地址,存储在地址寄存器中。$vertex\_id$ 为返回数据中储存的顶点编号。默认每个顶点需要 4 bytes 的存储空间。需要说明的是,每个缓存行大小为 64 bytes,对于无权重图每个顶点编号占用 4 bytes 的存储空间。在有权重图中每条边的权重和邻顶点编号组合存储需占用 8 bytes 的存储空间。所以每个缓存行可以存储 8 或 16 个顶点编号,最多生成 16 项预取请求。具体的编号大小可以在程序运行前配置到预取器中,用于判断如何读取顶点编号。

预取地址生成后,还需要对所有的预取地址进行比较,合并重复的预取地址。同时由于偏移数组返回的数据中包含每个顶点对应的边数组的起始和终止地址,对于边数组中超过当前终止地址的数据不进行预取,以减少错误预取的发射,提升预取准确率。

(2) 猜测预取。在含有较多社区结构的图数据集中,由于对顶点数组的间接预取发出的缓存请求数量较少,不能充分利用内存带宽,导致预取及时性较差。所以为了提升图数据预取性能,本文提出了顶点数组的猜测预取。

假设取回边数组数据中包含的邻顶点编号为{V0, V1,…,V15}。基于上一节的分析结果,对于不属于社区的顶点,上述邻顶点分布存储在最多 16 个不同缓存行内。而对于属于社区的顶点,上述顶点很可能存储在少量缓存行内。为了判断当前邻顶点是否密集存储,GSP 预取器计算了 V15 与 V0 之间的差值 D(邻顶点一般按编号大小顺序存储,V15 为当前最大编号值,而 V0 为最小编号值)。当其差值小于 48 时,说明当前邻顶点最多分布在 3 个缓存行内,预取器判定当前邻顶点密集存储在内存中,生成对顶点数组的猜测预取。而当差值大于等于 48 时,预取器认为当前邻顶点稀疏分布,不进行猜测预取。

为了提升预取的及时性,并充分利用带宽,猜测预取生成时,要考虑到 D 值的大小。设顶点 V15 的间接预取地址为 Paddr,则生成的猜测预取地址与 D 值范围关系如表 1 所示。对于不同的 D 值,基于返回的边数据,猜测预取都保证了预取器每次至少发出 4 项对顶点数组的预取。在模拟器中的实验结果表明,该数值可以较为充分地利用当前处理器的带宽,并保证顶点数组预取的及时性。

表 1　猜测预取地址生成

D 值范围	预取地址		
$D = 15$	Paddr + 64	Paddr + 124	Paddr + 192
$32 > D >= 16$	Paddr + 64	Paddr + 128	
$48 > D >= 32$	Paddr + 64		
$D >= 48$	不产生预取		

同样的,如果边数组返回地址超出当前处理顶点的边界,则不发出猜测预取。同时由于猜测预取存在一定的不确定性,所以最终的预取数据将存入二级缓存中,减少对一级数据缓存的污染。需要说明的是,对顶点数据进行猜测预取的过程是自动识别的,不需要预先识别当前处理图数据集的性质,可以在含有较多社区的图数据集上实现显著的性能提升。同时对于其他图数据集,猜测预取能够识别其中存在的密集存储的邻顶点,并进行预取,而对稀疏存储的邻顶点则不生成猜测预取。

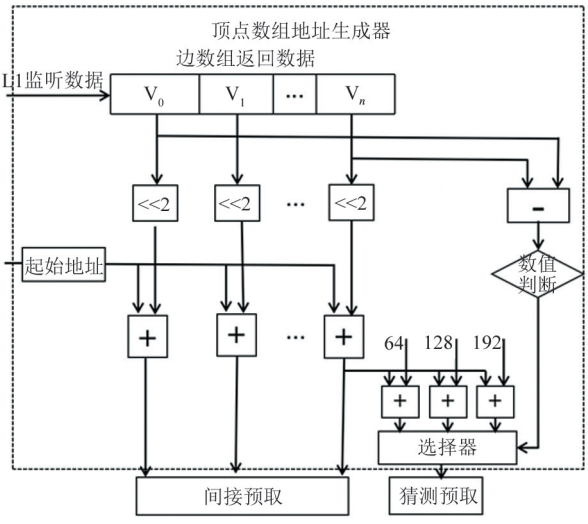

图 5　顶点数组地址生成器

3.4　边数组地址预取

如图 6 所示,边数组的预取也分成两部分,分别为间接预取和流式预取。预取器首先监听核内发往一级数据缓存的请求,当其地址属于偏移数组时,将该地址对应的物理地址记录到预取地址生成器中。然后当监听到该地址对应的数据返回到一级数据缓存时,将数据存入到边数组地址生成器中。上述数据存储的是每个顶点对应边数组的起始地址偏移值,通过之前记录的偏移数组访问地址筛选出当前处理顶点对应的偏移地址值。并根据图数据地址寄存器中的边数组的起始地址计算出边数组间接预取地址值,设预取地址为 PE, 则:

$$PE = offset + edge\_begin$$

其中 $offset$ 为偏移数组返回的偏移地址值,$edge\_beign$ 为预取器中存储的边数组起始地址。可以看出,与顶点数组的间接预取不同的是,边数组的间接预取只生成一项,而不会对返回的数据中所有的偏移值进行预取。这是因为在很多图算法中,不是所有顶点都需要进行运算的,所以边数组地址生成器记录当前处理顶点对应的偏移数组地址,用于筛选需要使用的偏移地址值。

理论上来看,偏移数组返回的地址中包含每个顶点对应的边数组的起始地址和终止地址,因而可以通过间接预取实现对当前处理顶点对应的全部边数据的预取。但是如果按照上述模式进行间接预取,则部分边数据过早地被存入一级数据缓存中,很可能在其被访问时,已经被替换出缓存,导致预取失效。为了解决上述问题,GSP 预取器只对每个顶点对应边数组的起始地址实现间接预取,因为该地址无法通过其他方式准确预测。而对于剩余的边数组地址,GSP 采用流式预取方式,当监听到来自核内的访存地址属于边数组时,发出的预取地址为该地址加 256,即该地址后面第 4 个缓存行的地址。这样做的目的是保证预取的及时性,使得发出的预取能够在真正请求到来前取回一级数据缓存。通常的流式预取中存在启动开销,即每个访存流的前几项不能及时被取回。为了缓解上述问题,GSP 预取器在生成对边数组的间接预取的同时,完成对间接预取地址后 3 个缓存行的预取,一定程度上减少了每个顶点对应边数组预取的启动开销。

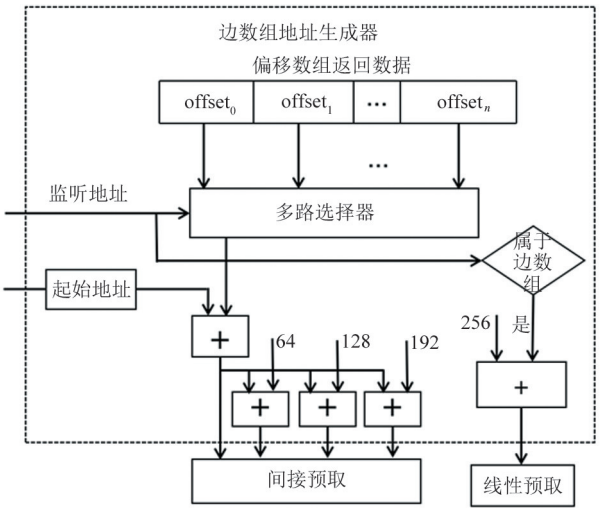

图 6　边数组地址生成器

3.5　预取器设计开销

为了评估预取器的设计开销,本文使用 Verilog 编程语言在龙芯处理器核内完成了 GSP 预取器的寄存器传输级(RTL)实现,并在 28 nm 工艺下对该设计进行了面积与功耗评估,处理器运行频率为 2.5 GHz。结果如表 2 所示,GSP 预取器占核内总面积的 0.16%,运行功耗占核内总功耗的 0.23%。

表 2　预取器面积功耗统计

预取器	面积/mm^2	面积占比	功耗/mW	功耗占比
GSP	0.088	0.16%	73	0.23%

由于 DROPLET 预取器没有进行精确的 RTL 级面积与时序评估。本文将从存储容量角度进行对比,GSP 预取器储存开销主要来自缓存队列,用于缓存预取的虚拟地址,占用 1 kB 的储存空间(主要来自 128 项 64 bits 的寄存器),而 DROPLET 预取器共使用了 7.7 kB 的储存空间。这是由于 DROPLET 预取器需要在内存控制处添加 TLB 进行虚实地址转换,而 GSP 预取器通过直接访问核内 DTLB 节省了上述存储开销。

4　实验结果与分析

本节将介绍实验所用数据和平台,展示最终的

实验结果并进行分析。

4.1　测试输入

为了验证预取器的性能表现,本文选取了 GAP 测试集内的 5 种常见图算法。如表 3 所示,其中 PR 算法为全部顶点参与计算的迭代算法,相对于其他算法访存规律性更强。其余算法顶点处理较为随机,间接预取可以实现更高的性能提升。而 SSSP 算法需要在带权重的图数据集上运行。

表 3　图算法描述

图算法	描述
PageRank(PR)	一种根据相邻顶点信息来计算顶点排名的迭代算法,用于网页排名的计算
Betweenness Centrality(BC)	一种计算顶点中心性的算法,顶点的中心性为该节点出现在其他两节点之间的最短路径上的比率[22]
Connected Components(CC)	一种计算图中所有的连通分量的算法,连通分量是图的一个子图,子图中任意两点之间均存在可达路径[23]
Breadth First Search (BFS)	以宽度优先顺序对图进行遍历的算法[24]
Single Source Shortest Paths (SSSP)	计算某个顶点到图中其他顶点最短路径的算法[25]

表 4 展示了本文选取的 4 种不同类型的真实图数据集,均来自社交网络和网页数据。顶点规模范围为 0.68 M ~ 22 M,平均度数范围为 11 ~ 105。由于模拟器上运行时间的限制,实验中没有选取较大规模的图数据集。但本文的预取器设计是基于图数据邻节点的存储特点,与图数据规模无关。且预取数据最终放入一级或二级缓存内,而选取的图数据规模皆大于二者容量,可以验证性能提升。同时本文的实验结论将同样适用于大规模图数据集。

表 4　图数据集描述

数据集	顶点数	边数	平均度数
Berkstan[26]	0.68 M	7 M	11
Hollywood[27]	1 M	112 M	105
Indochina[27]	7 M	191 M	25
Arabic[27]	22 M	631 M	27

4.2　测试平台

本文使用的模拟器为 sniper[28],是一款事件级精确的 CPU 模拟器。相对其他周期级精确的模拟器,sniper 在保证一定模拟精度的前提下,能够实现更快的模拟速度,适合模拟运行时间较长的图计算应用。模拟器的配置如表 5 所示,采用四核心结构模拟图计算并行运行情况,同时将算法代码中关键部分标记为热点区域,以缩短总的模拟时间。对于热点区域,模拟时开启 detail 模式,精确模拟处理器各部分运行状态。程序进入热点区域前运行在 cache warm-up 模式下,在热点区域模拟开始前对缓存进行预热,同时本文选取 GAP 作为图计算的软件框架。相对于其他复杂框架如 ligra 等,GAP 针对处理器进行的优化更少,可以较好地反映图计算运行时的真正瓶颈,便于验证预取器改进对图计算性能的影响。

表 5　模拟器配置信息

部件	配置
core	4 核心、32 项 load 队列、48 项 store 队列、128 项 ROB(reoder buffer)、发射与提交宽度为 4、处理器频率为 2.66 GHz
caches	三级结构、各级间关系为 inclusive、替换策略为 LRU、缓存行大小 64 B
L1 dcache	容量 32 kB、8 路组相连、数据访问时间 4 拍、tag 访问时间 1 拍
L2 cache	私有缓存、容量 256 kB、8 路组相连、数据访问时间 8 拍、tag 访问时间 3 拍
LLC	共享缓存、容量 8 MB、16 路组相连、数据访问时间 30 拍、tag 访问时间 10 拍
Memory	DDR3、内存访问时间 45 ns、带宽 25.6 GB · s^{-1}

4.3　性能对比与分析

为了验证 GSP 预取器的性能提升效果,本文选取了几种预取器与其进行了性能对比,如表 6 所示。其中 TGP(traditional graph prefetcher)为传统的图数据预取器,其设计理念与最新的 DROPLET 预取器相同,对顶点数组采用间接预取,对边数组采用流式预取。但预取位置并非在内存控制器处,因为该设计需要添加额外的TLB,且要在共享缓存处增加一

表 6 实验预取器介绍

预取器名称	特点	数据存放
GHB	基于缓存步长历史对当前访存流进行预取	二级私有缓存
Stream	识别等步长访存流并进行预取	二级私有缓存
TGP	图数据预取器,对顶点数组采用间接预取,对边数组进行流式预取	一级数据缓存
GSP	图数据预取器,对顶点数组进行猜测预取和间接预取,对边数组进行间接预取和流式预取	一级数据缓存和二级私有缓存

致性判断,较难在实际处理器中实现。因而本文将该预取器放置在一级缓存处,更好地与本文预取器进行对比,验证 GSP 预取器的相对性能提升。

图 7 为上述预取器相对无预取情况的性能提升比例,在 5 种图算法和 4 种图数据集共 20 个测试点中,本文预取器都实现了最高的性能提升。

表 7 为各类预取器在不同图算法下的平均性能提升,其中 GHB 预取器性能表现最差,相对无预取情况性能提升最低,在 PR 算法上甚至产生了性能下降。这是由于 GHB 预取器对整个访存历史进行缓存,不能识别图数据中3个数组访存规律的差异。

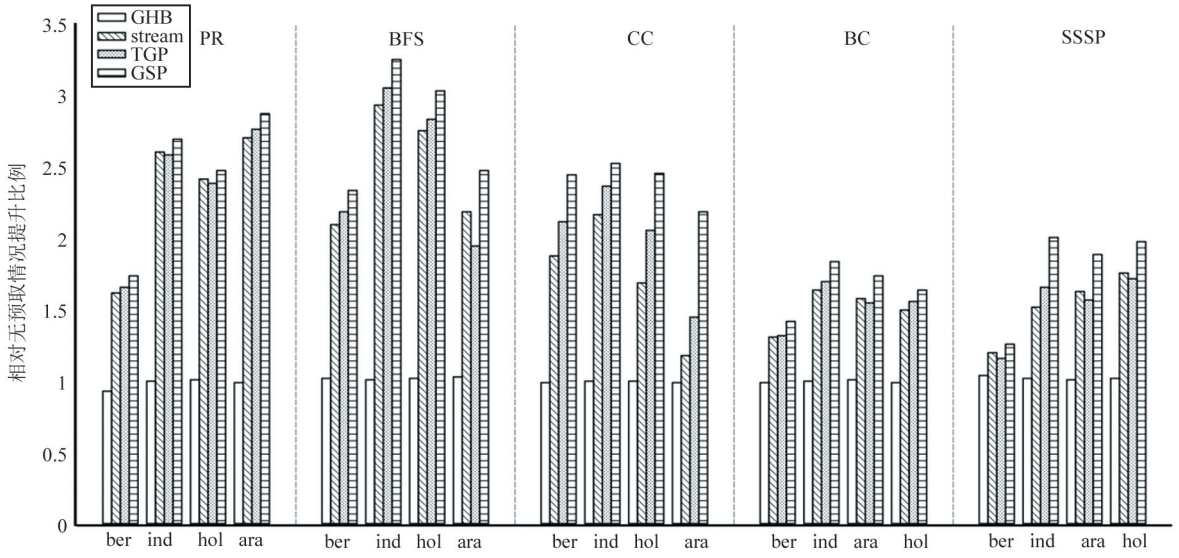

图 7 性能提升比例统计结果

表 7 不同算法平均性能提升

预取器	PR	BFS	CC	BC	SSSP
GHB	0.98	1.02	1.00	1.05	1.02
Stream	2.29	2.47	1.69	1.50	1.51
TGP	2.31	2.47	1.97	1.53	1.51
GSP	2.41	2.76	2.40	1.65	1.76

但图计算全局访存行为是无规律的,导致 GHB 预取器无法在图应用上取得性能提升。而 stream 预取器能够实现对不同地址范围的访存流的同时预取,因而可以识别 CSR 格式中的 3 个数组,实现了对边数组部分地址的准确预取。同时在社区中,由于邻顶点的密集存储,部分情况下,stream 预取器可以成功预取顶点数组的访存地址,最终实现了较高的性能提升。TGP 预取器的性能提升高于 stream 预取器。TGP 预取器能够利用访存数据准确预取顶点数组的访存地址,预取准确性高于 stream 预取器。但由于 TGP 预取器需等到访存数据返回才能发出间接预取,导致部分预取不能及时取回到相应缓存,预取及时性弱于 stream 预取器。

而本文提出的 GSP 预取器在 5 种图算法上皆实现了最高的性能提升,相对于 TGP 预取器的性能提升分别为 4.23%(PR)、9.72%(BFS)、18.08%(CC)、7.74%(BC)、15.4%(SSSP)。产生该性能提升有 2 个原因:一是 GSP 实现了对顶点数组的猜测预取,提升了该类数据的预取及时性;二是 GSP 对

边数组进行了间接与流式混合预取,相比 TGP 进一步提升了边数组预取的准确性和覆盖率。从图算法的角度看,GSP 在 PR 算法上性能相对提升最小,因为 PR 算法访存行为规律性最强,边数组的访存接近流式访存。顶点数组顺序处理,随机性最弱,因而 GSP 预取器相对提升最小。而在其他算法中,由于顶点处理具有较强的随机性,导致边数组和顶点数组的访问不规律,GSP 预取器能更好地发挥自身的优势。

4.4　预取及时性与准确性

评价预取器的性能有 3 个指标,即准确率、及时率和覆盖率。其中准确率是命中的预取数量与发出的预取数量的比值,准确率越高表明预取器发出的错误预取越少,对内存带宽浪费越小。而及时率是准确发出预取中已经取回到对应缓存的比率,及时率越高则真实请求命中预取后等待的时间越短,处理器内数据供应得更快。覆盖率则是预取器发出的预取数量占程序中数据能够被预取数量的比率,覆盖率越高表明预取器的算法越准确。但对图数据进行间接预取时,对于占较大访存比例的顶点数组,其预取覆盖率为 100%。所以在对预取器进行性能分析时,本文不对覆盖率进行统计。

将 TGP 预取器和 GSP 预取器在 5 个图算法上的预取准确率进行了对比。结果如图 8 所示,TGP 预取器在所有测试点上的准确率皆高于 GSP 预取器,这是由于 GSP 预取器相对 TGP 预取器对边数组进行了间接预取,对顶点数据进行了猜测预取。后者由于是猜测预取,会导致部分错误预取请求的发出,降低了 GSP 预取器的准确率。因而 GSP 预取器将顶点数组的猜测预取放入二级缓存处,一定程度地降低了错误预取带来的性能损失。从图算法的角度看,GSP 在 5 类图算法上的平均准确率分别为 85.22%(PR)、82.36%(BFS)、74.16%(CC)、38.05%(BC)、74.46%(SSSP)。对 PR 算法的预取准确率最高,因为 PR 算法对顶点进行顺序处理,随机性最小。而 BC 算法的预取准确率最低,该算法对顶点的处理顺序较为随机,边数组和顶点数组的访存规律性较弱。

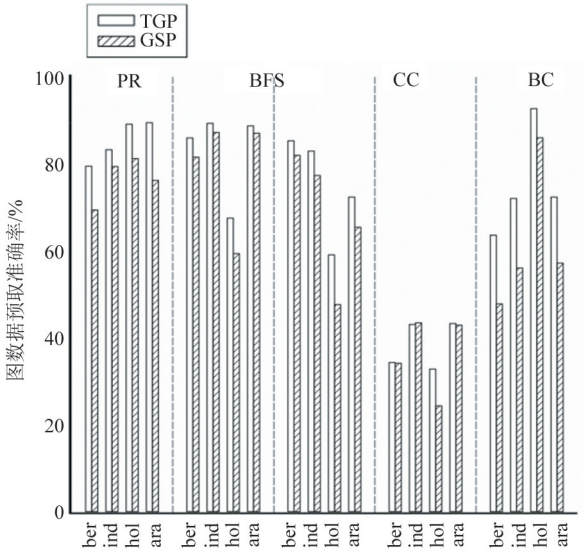

图 8　预取准确率统计结果

图 9 为 2 类预取器预取及时性的统计结果。可以看到,在所有测试点处,GSP 预取器的预取及时率皆高于 TGP 预取器。GSP 预取器通过对顶点数组进行猜测预取,有效利用了图数据中邻顶点的存储规律,及时发出了对顶点数组的预取。大幅度缩减了访存延迟,是 GSP 预取器相对于 TGP 预取器产生性能提升的主要原因。

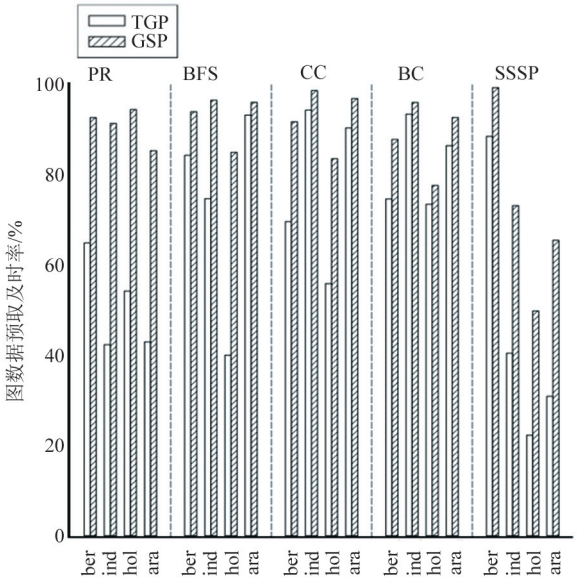

图 9　预取及时率统计结果

5　结　论

本文通过对不同类型图数据集储存规律的研

究,发现了图数据集社区结构中邻顶点紧密存储的特点,进而提出了利用该特性的图数据预取器设计方案。通过实现顶点数组的猜测预取,大幅提升了图数据预取的及时性,同时增加了对边数组的间接预取,添加了顶点数组间接预取发出的限制条件,提升了图数据预取的准确率和覆盖率。最终在 5 种图算法和 4 类图数据集共 20 个测试点上对本文提出的预取器进行了测试。实验结果表明,本文提出的预取器相对于无预取情况实现了 65% ~176% 的性能提升,相对于传统图数据预取器实现了 4.32% ~18.08% 的性能提升。

参考文献

[1] 张承龙, 曹华伟, 王国波, 等. 面向高通量计算机的图算法优化技术[J]. 计算机研究与发展, 2020, 57(6):1152

[2] KALAVRI V, VLASSOV V, HARIDI S. High-level programming abstractions for distributed graph processing [J]. *IEEE Transactions on Knowledge and Data Engineering*, 2017, 30(2):305-324

[3] KYROLA A, BLELLOCH G, GUESTRIN C. Graphchi: large-scale graph computation on just a PC[C]// Proceedings of 10th Symposium on Operating Systems Design and Implementation, Hollywood, USA, 2012:31-46

[4] HAN W S, LEE S, PARK K, et al. TurboGraph: a fast parallel graph engine handling billion-scale graphs in a single PC[C]// Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, USA, 2013:77-85

[5] 严明玉, 李涵, 邓磊, 等. 图计算加速架构综述[J]. 计算机研究与发展, 2021, 58(4):862

[6] FALSAFI B, WENISCH T F. A primer on hardware prefetching[J]. *Synthesis Lectures on Computer Architecture*, 2014, 9(1):1-67

[7] NESBIT K J, SMITH J E. Data cache prefetching using a global history buffer[C]// Proceedings of 10th International Symposium on High Performance Computer Architecture (HPCA'04), Madrid, Spain, 2004:96-96

[8] SHEVGOOR M, KOLADIYA S, BALASUBRAMONIAN R, et al. Efficiently prefetching complex address patterns [C]// Proceedings of 2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Waikiki, USA, 2015:141-152

[9] COOKSEY R, JOURDAN S, GRUNWALD D. A stateless, content-directed data prefetching mechanism[J]. *ACM SIGPLAN Notices*, 2002, 37(10):279-290

[10] ROTH A, MOSHOVOS A, SOHI G S. Dependence based prefetching for linked data structures[C]// Proceedings of the 8th International Conference on Architectural Supportfor Programming Languages and Operating Systems, San Jose, USA, 1998:115-126

[11] ZHANG C, ZENG Y, SHALF J, et al. RnR: a software-assisted record-and-replay hardware prefetcher[C]// 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Athens, Greece,2020: 609-621

[12] YU X, HUGHES C J, SATISH N, et al. IMP: indirect memory prefetcher[C]// Proceedings of the 48th International Symposium on Microarchitecture, Waikiki, USA, 2015:178-190

[13] AINSWORTH S, JONES T M. Graph prefetching using data structure knowledge[C]// Proceedings of the 2016 International Conference on Supercomputing, Istanbul, Turkey, 2016:1-11

[14] BASAK A, LI S, HU X, et al. Analysis and optimization of the memory hierarchy for graph processing workloads [C]// Proceedings of 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA), Washington, USA, 2019:373-386

[15] SHUN J, BLELLOCH G E. Ligra: a lightweight graph processing framework for shared memory[C]// Proceedings of the 18th ACM SIGPLAN Symposium on Principlesand Practice of Parallel Programming, Shenzhen, China, 2013:135-146

[16] GIRVAN M, NEWMAN M E J. Community structure in social and biological networks[J]. *Proceedings of the National Academy of Sciences*, 2002, 99(12):7821-7826

[17] FORTUNATO S. Community detection in graphs[J]. *Physics Reports*, 2010, 486(3-5):75-174

[18] 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, Beijing,China, 2008:695-704

[19] HUANG B, LIU Z, WU K. Structure preserved graph re-

ordering for fast graph processing without the pain[C]// 2020 IEEE 22nd International Conference on High Performance Computing and Communications, Yanuca Island, Fiji, 2020:44-51

[20] ZHANG Y, LIAO X, JIN H, et al. FBSGraph: accelerating asynchronous graph processing via forward and backward sweeping[J]. *IEEE Transactions on Knowledge and Data Engineering*, 2017, 30(5):895-907

[21] MUKKARA A, BECKMANN N, ABEYDEERA M, et al. Exploiting locality in graph analytics through hardware-accelerated traversal scheduling[C]// Proceedings of 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture(MICRO), Fukuoka, Japan, 2018:1-14

[22] BRANDES U. A faster algorithm for betweenness centrality[J]. *Journal of Mathematical Sociology*, 2001,25(2): 163-177

[23] SUTTON M, BEN-NUN T, BARAK A. Optimizing parallel graph connectivity computation via subgraph sampling [C]// Proceedings of 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), Vancouver, Canada, 2018:12-21

[24] BEAMER S, ASANOVIC K, PATTERSON D. Direction-optimizing breadth-first search[C]// Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, Salt Lake City, USA, 2012:1-10

[25] MEYER U, SANDERS P. Δ-stepping: a parallelizable shortest path algorithm[J]. *Journal of Algorithms*, 2003, 49(1):114-152

[26] LESKOVEC J, SOSIČ R. Snap: a general-purpose network analysis and graph-mining library[J]. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2016, 8(1):1-20

[27] DAVIS T A, HU Y. The University of Florida sparse matrix collection[J]. *ACM Transactions on Mathematical Software(TOMS)*, 2011, 38(1):1-25

[28] HEIRMAN W, CARLSON T, EECKHOUT L. Sniper: scalable and accurate parallel multi-core simulation[C]// Proceedings of 8th International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems (ACACES-2012), Fiuggi, Italy, 2012:91-94

Design of graph data prefetcher based on community structure

LI Ce, ZHANG Longbing

(State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

(School of Computer Engineering, University of Chinese Academy of Sciences, Beijing 100190)

Abstract

Due to the large scale and irregular structure of graph data, a large number of high-latency memory accesses are generated when graph applications are running, which greatly reduces the efficiency of general-purpose processors. This paper uses a combination of software and hardware to design a dedicated prefetcher for graph analytics. Using the characteristics of graph data access and the storage law of community structure, and through hybrid prefetching of graph data, the memory access latency of graph analytics are shortened and significant performance gains are obtained on graph datasets containing more communities. Experiments on different graph algorithms and graph datasets show that the prefetcher achieves 65% – 176% performance improvement over the no-prefetch baseline, 6% – 21% performance improvement over the stream prefetcher, and 4% – 18% performance improvement over the traditional graph data prefetcher.

Key words: graph analytics, prefetcher, community structure, storage regular pattern, timeliness