高技术通讯 2024 年 第34 卷 第7 期:714-725

doi:10.3772/j.issn.1002-0470.2024.07.005

针对深度学习中不规则内存访问的高吞吐内存管理单元①

丁 峰② 李 曦③

(中国科学技术大学计算机科学与技术学院 合肥 230026)

摘 要 人工智能应用的多样化与复杂化导致了算法模型的不规则内存访问,即集中突发的访问请求与稀疏的访问地址,从而给智能应用在内存资源严格受限的移动端设备的部署带来了挑战。这种不规则的内存访问导致了现有架构中内存管理单元(MMU)的地址转换面临低吞吐和长延迟的问题,使其成为系统访存通路的瓶颈。针对上述问题,本文提出了一种新的高吞吐 MMU 架构方案(HTMMU),通过多流并行,加强冗余请求的过滤,合理地分配有限的片上存储资源等手段,从而能高吞吐、低延迟地处理不规则访问的地址转换,提升系统访存效率。实验结果表明,在处理人工智能算法内突发的稀疏访存时,相较于当前主流 MMU 设计方案,HTMMU 平均获得了 2.43 倍的性能提升,而平均访问延迟降低为原先的 34.1%,同时将额外面积开销控制在 3.0% 以内。

关键词 内存管理单元(MMU);地址转换;不规则访存;深度学习;高吞吐

人工智能应用的多样化与复杂化导致了深度学习模型中的不规则内存访问。这种不规则性体现在 2 个方面:(1)稀疏的访问地址;(2)集中突发的访问请求。一方面,稀疏的地址访问行为是人工智能算法中常见的运行特点之一。例如推荐系统[1]中用户—物品的交互矩阵,其本质是稀疏度极高(95% ~99%)的稀疏矩阵[2-3],即使用嵌入技术将用户和物品的特征转为低维、密集的嵌入向量,其嵌入表查找过程仍会受到输入数据的随机性影响而产生大量不规则访存;使用图神经网络(graph neural network,GNN)[4]处理真实世界的大规模图数据时,其非结构化稀疏度往往在 75% ~99% 之间[5],在推理和训练 GNN 时,聚合运算通常需要在稀疏图中访问所有邻接节点的信息,导致了大量稀疏访存;在使用剪枝技术压缩后的神经网络中,其输入激活矩阵(***IA***)和权重矩阵(***W***)都是稀疏矩阵[6-7]。另一方面,集中突发的访存请求是人工智能算法的另一个执行特点。通常,人工智能算法将数据以批次(batch)为单位进行推理或训练。虽然分批次的数据读取和执行可以最大化地利用硬件计算和带宽资源,但也容易导致某批次数据在短时间内集中访存的现象。

上述不规则的访存行为给移动端设备中的内存管理单元(memory management unit,MMU)带来了巨大的吞吐压力,进而导致系统访存效率下降。具体地,移动端设备中直接存储器访问(direct memory access,DMA)单元的访存效率直接影响了系统的执行速度。但由于移动端设备的片上资源严格受限,因此参与运算的多维张量无法被完整地放入片上内存[8]。于是,这些多维张量通常会被 DMA 单元拆解为多个位于片外内存的线性内存事务,再以串行的方式载入或卸出片上内存空间。当上述不规则的访存请求到来时,DMA 单元需要在短时间内产生大量内存事务来搬运被拆分的张量。由于这些内存事

① 国家自然科学基金(U20A20227,U22A2028),中国科学院稳定支持基础研究领域青年团队计划(YSBR-029)和中国科学院青年创新促进会资助项目。

② 男,1997 年生,硕士生;研究方向:计算机系统结构;E-mail:df12138@mail.ustc.edu.cn。

③ 通信作者,E-mail:llxx@ustc.edu.cn。

(收稿日期:2023-05-10)

务通常来自于不同的片外内存页面中,因此 MMU 的地址转换机制必须在 DMA 访问前确定这些内存事务的物理页面。然而,当前主流的 MMU 架构存在吞吐率不足的缺点,导致 DMA 单元无法及时地从 MMU 处获得物理页面,最终导致其无法有效地利用内存带宽,出现访存瓶颈,拖慢了系统运行效率。出现上述问题的核心原因是,主流的 MMU 架构过于注重利用数据的良好局部性,忽视了低吞吐率可能导致的性能问题[9]。在面对稀疏的访问地址和集中的访存行为时,主流 MMU 的转译后备缓冲区(translation lookaside buffer,TLB)既没有较高的命中率,也没有较高的吞吐率应对集中的访问行为。

基于上述观察,本文提出了一个基于内存资源受限场景下的高吞吐 MMU 设计方案(high-throughput MMU,HTMMU),该方案通过增加并行度,合理分配有限片上存储资源等手段提升了 MMU 的吞吐率,降低了地址转换延迟,从而提升系统访存效率。具体地,HTMMU 包括了 2 处关键设计:(1)在传统多层级 TLB 架构的基础上,采用多流设计并行地处理地址转换请求,并使用流路由单元均衡多流之间的负载,提升了 MMU 的吞吐率;(2)在传统 MMU 硬件页表遍历方案的基础上,加强了对冗余页表请求的过滤,提高页表遍历的实际有效带宽。实验结果表明,HTMMU 在处理不规则的集中突发地址转换请求时,相较于主流的 MMU 设计方案,HTMMU 以不到 3.0% 的额外面积开销,提供了 2.43 倍的性能提升,并将平均访问延迟降低到了原先的 34.1%。

1　研究背景

1.1　人工智能算法中的不规则访存

许多成熟且已广泛使用的人工智能算法中存在了大量的不规则访存行为,即稀疏的访存地址和集中突发的访存请求。

比如,推荐系统中常用的点击率预估算法(如模糊神经网络(fuzzy neural network, FNN)[10]、Wide&Deep[11]等)在对嵌入层进行推理和训练时会产生大量集中突发的稀疏访存请求。目前主流的点击率预估算法大多基于图 1 所示的经典“嵌入 + 多层感知机(embedding + MLP)”范式。使用该范式的模型通常由 3 部分组成,即嵌入层、特征交叉层和分类器。其中,嵌入层使用复杂的嵌入技术在计算机内表示用户(或物品)的非数值特征(比如商品产地、分类等,亦称稀疏特征)。

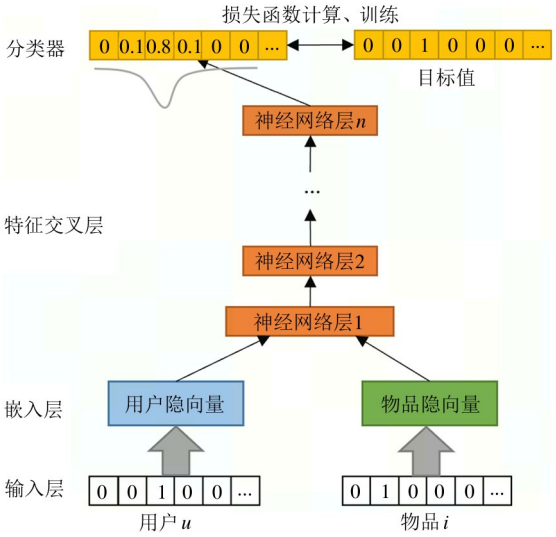

图 1　基于深度学习的点击率预估模型的经典结构[2]

图 2 描述了嵌入层推理的一般流程。哈希嵌入法是针对稀疏特征的常见嵌入方法[12],该方法首先使用独热码(one-hot code)编码物品的某个稀疏特征所有可能的类别,生成维度极大的独热向量,再根据物品的特征对生成的独热向量进行 0-1 赋值,集合所有独热向量后可组成一个甚稀疏的独热矩阵。然后,算法通过哈希函数将独热矩阵进行重映射,以减少矩阵的维度,最终获得索引矩阵 $\boldsymbol{I}$。嵌入表矩阵 $\boldsymbol{W}$ 可由嵌入层随机初始化,或者通过某种先验知识得到。嵌入表查找(embedding lookups)过程即是算法通过计算 $\boldsymbol{I}\times\boldsymbol{W}$ 获得嵌入向量 $\boldsymbol{Emb}$ 的过程,但由于矩阵 $\boldsymbol{I}$ 内部保存的数值对应了嵌入表矩阵 $\boldsymbol{W}$ 内嵌入向量的行号,因此嵌入表查找的本质是选择嵌入表矩阵中的向量。当嵌入层在处理大规模数据时,输入数据的随机性和用户—物品交互记录的稀疏性会导致嵌入表查找过程产生稀疏的访问地址;而当模型为提高计算效率,使用多设备模型并行[3]的方式执行嵌入表查找时,全对全(all-to-all)的通信模式又会在短时间内集中产生大量访存请求。

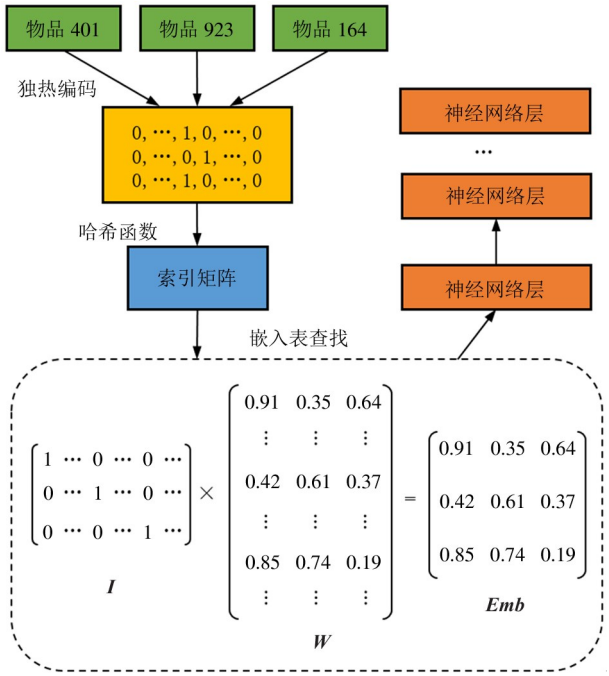

图 2　模型推理时嵌入表查找的主要过程

又比如,用于处理非欧氏空间数据的图神经网络(GNN)[13]在执行聚合运算时也会产生大量稀疏访存。大多 GNN[14]的图神经元的计算过程可总结为以下 3 步。

(1)对于每对邻居点 (x, y) 及其边 e_{xy},以上一层迭代的点和边特征向量作参数,使用消息函数计算得到消息向量 $\boldsymbol{m}_{xy}$。

(2)以每个节点 x 的特征向量和上一步得到的消息向量 $\boldsymbol{m}_{xy}$作参数,使用聚合操作计算得到向量 $\boldsymbol{s}_x = \sum \boldsymbol{m}_{xy}$。

(3)将得到的向量 $\boldsymbol{s}_x$ 和上一层的点特征向量作参数,使用更新函数计算得到下一层特征向量。

与边相关的计算是最主要的性能瓶颈[14]。例如,GCN[15]和门控图神经网络(gate graph sequence neural network,GGNN)[16]在聚合运算上消耗的时间占了上述 3 个计算步骤总时间的 40% ~ 50%[14]。因为对每条边执行消息传递函数以及对消息向量的散播和聚合都会产生大量稀疏访存。

1.2　典型 MMU 架构

MMU 是系统访存通路中不可或缺的重要组成部分。其主要功能有:完成虚实地址转换,实现进程间内存空间隔离;控制访问权限,保护操作系统及代码段的内存数据;形成统一的地址空间,方便智能应用数据在多个设备间搬运等。在处理器架构中,MMU 位于片上内存和片外内存之间,是处理器访问片外内存的关键路径上的节点,如图 3 所示。

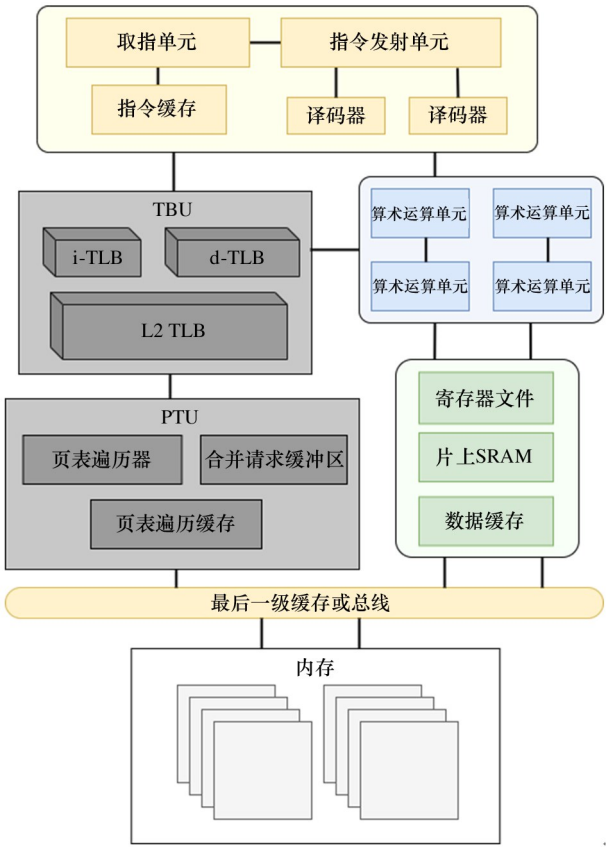

图 3　内存管理器在处理器中的典型位置

当前主流的 MMU 架构通常包括两大单元,即转译后备缓冲区单元(translation lookaside buffer unit,TBU)和页表遍历单元(page-table walker unit,PTU)。其中,TBU 内包含多个多层级的 TLB,一级 TLBs 会将指令和数据分开存储在指令 TLB(i-TLB)和数据 TLB(d-TLB)中[17],并分别适配不同的替换策略,以更好地利用指令和数据中相异的局部性,同时减少流水线冒险,加速地址转换;二级共享 TLB(L2 TLB)作为一级 TLB 的次级存储器,拥有更大的空间,可同时存放指令和数据,有些可同时支持不同的页大小[18],充分利用大页和小页各自的优势:大页可以明显地提升 TLB 的命中率,而小页可以提供更细粒度的页面保护和权限。x64 系统便可同时支持使用 4 kB 和 2 MB 页面,通过对最后一级页表的切换实现对页面大小的切换。

PTU 负责在 TLB 丢失后,用硬件执行页表遍历请求,最终寻得目标页物理地址。相较于传统的软件方法,使用硬件完成页表遍历更加迅速,通常只需要几十个时钟周期。PTU 主要由 3 部分组成,包括页表遍历器、合并请求缓冲区[9]和页表缓存[19]。其中页表遍历器负责控制每个页表遍历请求的状态和过程,其数量通常与合并缓冲区的条目数一致;合并请求缓冲区负责将目标物理页相同的请求合并,过滤冗余的页表遍历请求。具体操作是,合并请求缓冲区内的记分板(scoreboard)保存了正在执行页表遍历请求,而对应该记分板的多个合并项保存了与该请求的目标物理页相同的页表遍历请求。合并缓冲区在页表遍历开始前,通过比对当前请求的虚拟地址与所有记分板内请求的虚拟地址,识别出目标物理页相同的冗余请求,将它们置入对应记分板的合并项中,阻止它们进入真正的页表遍历阶段,从而实现冗余请求的过滤,缓解 PTU 的吞吐压力。当记分板内的请求完成页表遍历后,对应合并项内的请求也得到了需要的物理地址,完成了页表遍历;页表缓存[19]缓存了最近使用的地址转换路径,利用前几层页表的局部性加速页表遍历过程。

然而,主流的 MMU 架构重点关注数据的局部性,忽视了对 MMU 吞吐率的提升。例如,Intel Skylake 处理器[20]将大量片上资源留给了多层级的 TLB,以期尽可能地提升命中率,但在吞吐率方面仅能同时支持 4 个页表遍历请求,这在面对短时间集中突发的稀疏访存请求时是远远不足的。

2　问题与挑战

深度学习算法中的不规则的地址访问具体表现为如下两方面的问题。

2.1　突发的大量地址转换

短时间内突发的大量地址转换请求会给 MMU 带来巨大压力。图 4 是训练深度学习推荐模型(deep learning recommendation model,DLRM)时嵌入表查找时间占总时间的变化图,可见嵌入表查找在某几个时间点上出现了明显的峰值。这是因为处理器在运行深度学习算法时,往往需要处理 MB 级别的数据块搬运(涉及到张量卸载、预取,中间结果的保留与更新等)。假设处理器在执行卷积运算时,需要将 4 MB 的输入数据和权重从片外搬运至片内缓存中,采用 4 kB 页面粒度(如今大多系统常用的页面大小)管理内存,则此次数据搬运至少会涉及(4 MB/4 kB) = 1 000 个不同页面,再考虑到参与运算的数据大多为多维张量,而且这些数据大多是经过压缩后使用非结构化的数据结构表示,那么实际需要的页面会远多于该估计值。事实上,AlexNet、GoogleNet 和 ResNet 在最坏情况下需要最多 1 500 个不同的页面[9]。这对 MMU 处理地址转换请求的吞吐能力提出了更高的要求。

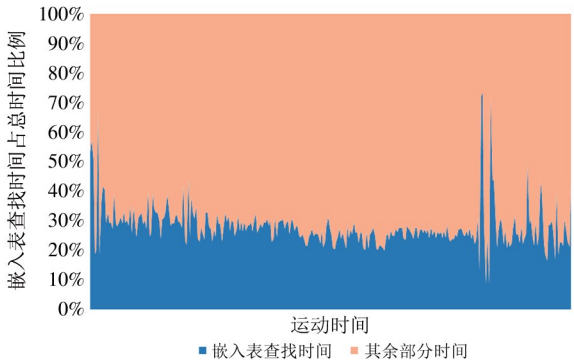

图 4　训练 DLRM 时嵌入表查找所消耗百分比时间

2.2　过于稀疏的访存行为

过于稀疏的访存行为加剧了地址转换中的性能损失。仍以 Criteo 数据集训练 DLRM 网络为例,该数据集中每行(项)记录有 26 个稀疏特征和 13 个稠密特征。假设嵌入向量的维度设置为 32,并使用 FP16 数据类型,那么表示一行记录需构建一个 26×32 的嵌入表,该批次数据(2 048 个)的所有嵌入向量会占用 1.6 MB 的内存空间,而这些嵌入向量却分布在几十甚至上百 MB 级别的地址范围内,整个数据集的嵌入向量会达到数十乃至上百个 GB。在嵌入表查找阶段,智能应用会试图从数十 kB 乃至数十 MB 级别的地址范围内寻找该批次数据中所有的嵌入向量,图 5 展示了这一阶段的访存跨度。

从图 5 的内存访问行为可知,使用 4 kB 页大小(图中黑实线表示一个 4 kB 页能覆盖的范围)时,一个页仅能覆盖当前内存访问的 61%,即便使用 2 MB

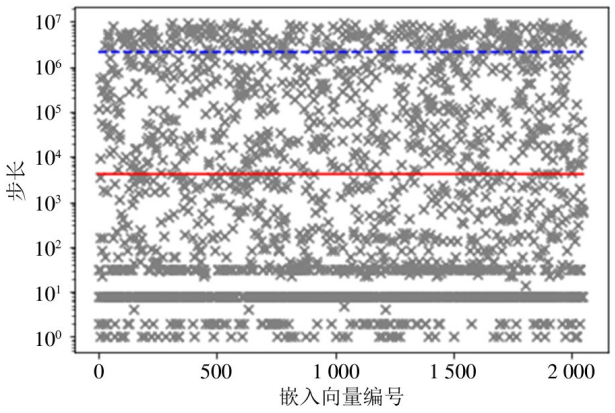

图 5　在 Criteo 数据集上训练 DLRM 时嵌入表查找的内存访问行为

的大页(图中黑虚线),也有超过 12.9% 的内存访问事务落在另一个页面上。这意味着当 TLB 为该批次数据预留出一项条目时,TLB 的命中率约为 87%,而在一般情况下,使用 4 kB 页面大小的 TLB 命中率也能达到 95% 以上,因此这部分稀疏访存请求导致的 TLB 未命中数量约是平时的 3 倍,这对页表遍历吞吐提出了更高的要求。

3　架构设计

3.1　整体设计

针对上节详细描述的集中突发的地址转换请求和过于稀疏的访存行为问题,本文提出了一个高吞吐 MMU 架构——HTMMU,其具体架构见图 6。在片上内存资源严格受限的场景下,HTMMU 以典型的 MMU 架构为基础,做了 2 点主要改进:(1)为扩大 MMU 的吞吐率,在 TBU 内使用多流设计并行地处理地址转换请求,并用流路由模块(streamId router)均衡多流之间的负载;同时通过多次实验探究,求得了多级 TLB 下的最优资源分配方案。(2)为充分过滤冗余的页表遍历请求,减轻页表遍历吞吐压力,折中地(tradeoff)增加了 PTU 的合并请求缓冲区的合并项数量,找到了面积开销与性能提升的平衡点。此外,对翻译路径缓存(translation path cache,TPC)作了适当地裁剪,给合并请求缓冲区增加的合并项腾出了部分片上内存资源。下面逐一详细介绍上述设计。

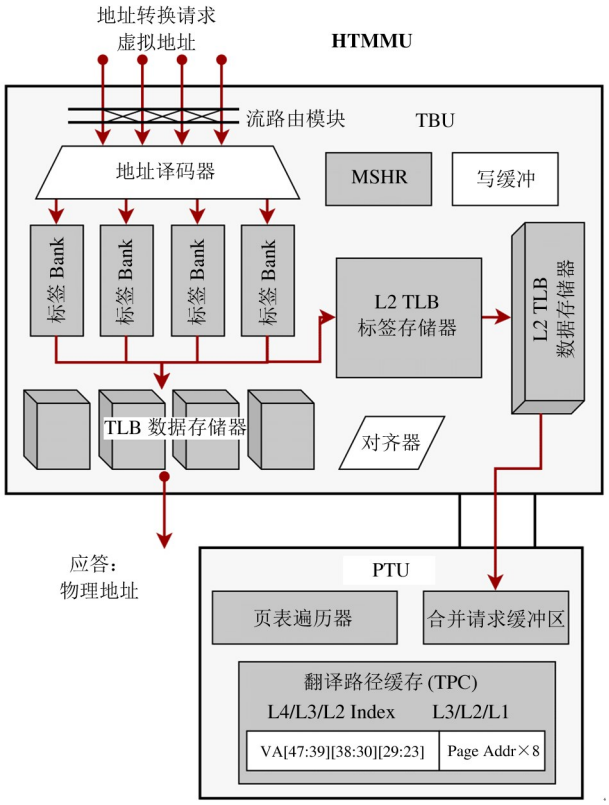

图 6　HTMMU 设计架构图

3.2　TBU 架构设计

为应对集中突发的大量地址转换,扩大 MMU 吞吐率,HTMMU 使用了多流方案:将原先输入到 MMU 的数据地址转换请求由单流拆分为多个并行的数据流,通过牺牲单块 TLB 的容量(即牺牲局部性)来换取吞吐量,让更多的地址转换请求能被并行执行;之后,为了在多个流之间实现负载均衡,在输入端口处添加了流路由模块来实时地调整每条流的流量。

方案的具体实现如下。多流方案将原先一级 TLB 中的 d-TLB 拆分成多个并行阵列(bank),并将 MMU 的数据地址转换请求端口扩展成多个,再添加上更多的逻辑电路,参见图 7,使得 MMU 在面积开销基本未变的情况下获得了更高的并行度。

流路由模块可通过分析每个 TLB 流的阻塞信息,实时地决定每个请求应使用哪个流完成地址转换,从而更好地均衡各个流的负载压力。相应地,在每个地址转换请求中添加了位域 Streamid,用来指示该地址转换请求的流 id。Streamid 的默认值是虚拟地址的高位,当流路由模块发现默认流受阻,而其

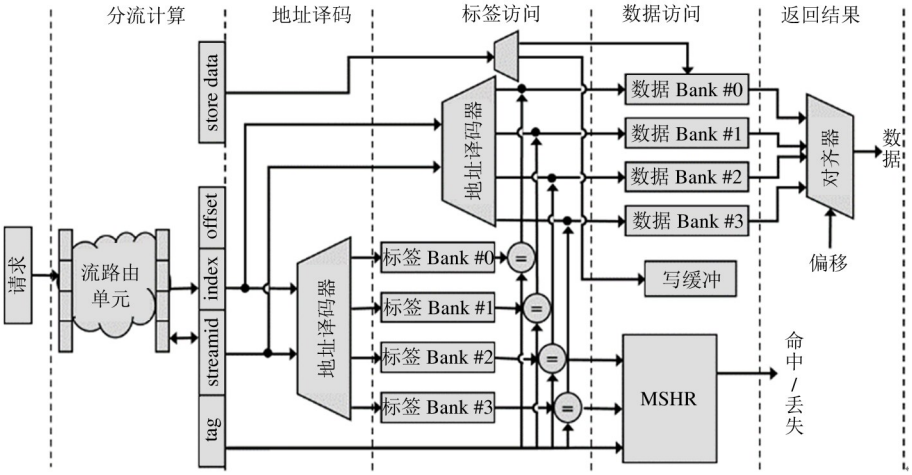

图 7　d-TLB 的数据通路设计图

他流未受阻时,会优先使用未受阻的流完成地址转换。

3.3　PTU 架构设计

PTU 负责在 TLB 丢失后,用硬件执行页表遍历,通过访问主存内的多级页表寻得目标页物理地址。而 PTU 内的合并请求缓冲区负责将目标物理页相同的请求合并,从而过滤掉冗余的页表遍历请求。

前文的分析已经指出,过于稀疏的访存请求会导致约 3 倍于一般情况的 TLB 丢失请求,这些请求会进入 PTU 进行页表遍历,从而加重了 PTU 的页表遍历负担。然而 18% ~44% 的页表遍历请求最终会落在同一个页面上,而其中被现有合并请求缓冲区过滤的数量不到 80%。

为解决合并请求缓冲区无法充分过滤冗余页表遍历请求的问题,HTMMU 在基准 MMU 架构的基础上,增加了合并缓冲区条目中的合并项数。图 8 展示了一个合并缓冲区条目的设计细节,图左侧的记分板存放了请求的虚拟地址、页表层级等重要信息,图右侧的合并项则记录了被合并的请求。增加了合并项数后,会有更多冗余的页表遍历延迟被阻拦,不进入 PTU 执行真正的页表遍历,缓解了 PTU 的吞吐压力。这些冗余请求的延迟仅取决于对应记分板中的请求:当记分板内的请求完成页表遍历后,对应合并项内的请求也得到了需要的物理地址,完成了页表遍历。

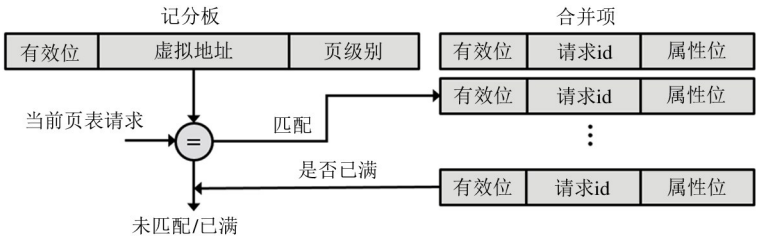

图 8　一项合并请求缓冲区的实现细节

然而,增加合并项数必然会占用更多的片上内存资源。4.3 节将通过多次实验,找到性能提升和面积开销的最佳平衡点。此外,本研究发现适当减少 TPC 的项数对性能和延迟的影响很小。因此,在内存资源不足时,可以通过裁剪 TPC 的项数来节省片上资源。

4　实　验

4.1　测试用例及基准方案

本文构造了 3 种不同的测试集来衡量方案性能:

(1) 密集型访存行为(记为 Dense)。主要包括

了一般情况下片上内存和片外内存之间的数据搬运操作。

（2）较稀疏型访存行为（记为 Big）。主要包括了片外内存上较稀疏的数据访问，步长通常在 1 GB 以内。

（3）稀疏型访存行为（记为 Large）。主要包括了极端情况下非常稀疏的数据访问，有些步长会超过 1 GB。

上述测试集合中，Dense 和 Large 部分测试用例是随机生成的，而 Big 部分稀疏测试用例来自使用 Criteo 数据集训练 DLRM[3]、deepFM[21] 时嵌入层的内存访问行为，以及来自使用 MTWM 数据集训练知识图卷积神经网络（knowledge graph convolutional network，KGCN）[13] 的访存行为。

用于比较实验方案性能的基准 MMU 是一种目前移动端处理器上使用的典型 MMU 架构方案。参考图 3 中 MMU 架构，除了在 1.3 节中详细介绍的架构设计外，该基准 MMU 的 TLB 还具有如下 3 个 属性。

（1）丢失时命中。TLB 内带有丢失状态处理寄存器（miss status handling register，MSHR）单元，可保留未命中的 TLB 请求，从而重叠未命中延迟。

（2）伪 LRU 替换策略，平衡了硬件实现成本和性能

（3）结合条目[22]，一个缓存块中可存储多个连续的物理地址。

基准模型其他详细参数细节参考表 1。

表 1　基准 MMU 的模型参数表

相关组件	参数及说明
i-TLB	2 组 16 路组相连，每项 64 字节，7 周期命中时间
d-TLB	16 组 4 路组相联，共 64 项，每项 32 字节，4 ~5 周期命中时间
L2 TLB	16 路组相联，共 512 项，每项 32 字节，7 周期命中时间
PTU	16 项缓冲区，每项 4 个合并项
TPC	全相联页表遍历缓存，每项 48 字节，8 ~9 周期命中时间
页表	4 kB 或 2 MB 页大小，四级或三级页表（实验中使用 2 MB）
片外内存	固定 1 000 周期访问时间

本实验使用 2 个性能指标来衡量 MMU 的性能和吞吐率，即总执行时间和请求的平均等待时间。总执行时间指 MMU 完成所有的地址转换请求所需的时间，反映了 MMU 的性能。请求平均等待时间指所有请求自发出到完成地址转换所需的平均时长，在测试样例不变的前提下，更短的平均等待时长表明 MMU 在单位时间内处理的平均请求数更多，因此可反映 MMU 的吞吐率。由于总执行时间受到最后一个地址转换请求发出时刻的影响，因此请求平均等待时间更能反映出 MMU 的吞吐率，而总执行时间的缩短能间接证明本文解决了吞吐率不足导致的性能问题。另需注意，4.3 节中所有的性能（总执行时间的倒数）和延迟（请求平均等待时间）数据都已经过归一化处理。

4.2　实验平台

以当前学界广泛使用的 gem5[23-24] 模拟器为平台展开实验。因为相较于物理上实现架构方案，使用软件模拟器建立模型更加灵活快速。本实验在 gem5 模拟器基础上，添加了特定指令集体系结构（instruction set architecture，ISA）和相关微指令编码，使 gem5 在全系统模式下（full-system mode）能执行与稀疏访存相关的指令和微指令，并在 gem5 上构建了一套基本的地址转换系统模型，见图 9。

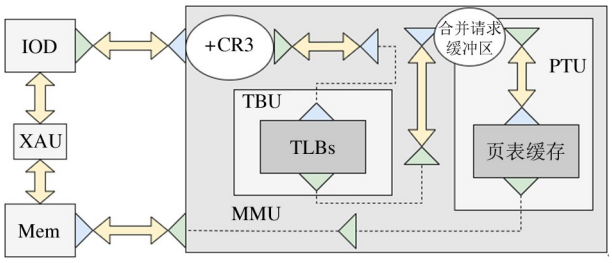

图 9　gem5 中搭建的地址转换子系统模型示意图

指令译码模块（I/O instruction decoder，IOD）负责指令读取和译码，它会将访存指令拆分出大量的微指令（μop），并将指向片外内存的虚拟基地址和偏移量等信息发送到 MMU 中进行地址转换，访问片上内存无需执行地址转换。IOD 得到物理地址后，会通过执行模块（execute arithmetic unit，XAU）进一步执行指令功能。

MMU 负责对接收来的虚拟地址作地址转换。

该模型工作流程如下所述。(1)将收到的虚拟地址与存放页目录物理地址的寄存器 CR3 相加,并抛弃低位的页偏移位,得到虚拟页号(virtual page number,VPN)。(2)将 VPN 发送到 TLB 中,查找 TLB 是否已缓存了相应的物理地址,若存在则地址转换完成,若不存在则进行下一步。(3)VPN 被发送到页表遍历单元中,经过若干次访问页目录后找到对应的物理地址。(4)VPN 和物理地址会被缓存到 TLB 中,并发回给 IOD。

IOD 收到物理地址后,会将微指令和物理地址一同发给 XAU,XAU 会完成后续的数据访存、运算等具体任务。这些内容不属于本文讨论的重点,本文对这部分任务做了合理的抽象。

4.3　实验结果

在 TBU 架构设计中,多流方案可以并行地处理多个地址转换请求,从而达到数倍于原设计的峰值吞吐率。然而,在保持片上存储资源不变的前提下,更多的 bank 意味着每个 bank 只能获得更少的容量,导致更多的冲突丢失(conflict miss)和更高的丢失率,从而抵消了扩大并行度后的优化效果。因此,继续增加 bank 数量无法获得成比例的吞吐率和性能提升。图 10 展示了使用 2、4 和 8 个流 bank 时的性能提升和延迟改善情况。其中,4 流设计分别在 Big 和 Large 测试集上获得了平均 2.08 倍和 3.42 倍的性能提升;延迟方面,每个请求的平均等待延迟降低到了原先的 41% 和 44%。而使用 8 流设计时没有获得成比例的性能提升和延迟收益,考虑到更

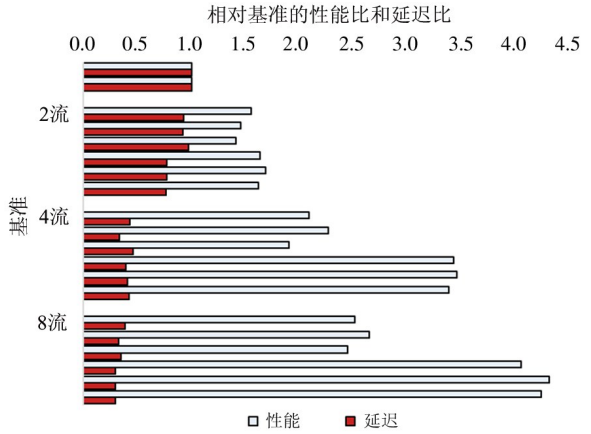

图 10　2 流、4 流和 8 流设计在 Big 和 Large 集的性能提升与延迟改善情况

多的 bank 会增加更多的逻辑电路开销,因此本文折中地将 4 流设计作为最终方案。

此外,本文在 4 流方案的基础上进行了多次实验,找到了提升性能的最优资源分配方式,见图 11。实验发现,在保持 TBU 的片上存储资源不变的情况下,使用更小的一级 TLB(16 项和 32 项)可获得相较于最差方案 5% ~10% 的性能提升,并将延迟降低至最差方案的 60% ~90%。这是因为使用多流方案后,一级 TLB 未命中数平均增加了 31.8%,因此需要扩大二级 TLB 的容量,以捕获更多一级 TLB 未命中请求,提升性能。

综合分析图 10 和图 11 的数据,选择 32 项一级 TLB 和 544 项二级 TLB,以及 4 流并行的 TBU 设计方案。相比基准方案,HTMMU TBU 在执行 Big 测试集(取自 DLRM、deepFM 和 KGCN 网络的部分访存行为)时,获得了平均性能 2.48 倍的提升,平均延迟降低至原先的 30.0%,在 45 nm 工艺下,保守估计增加了 3.1% 的面积开销。

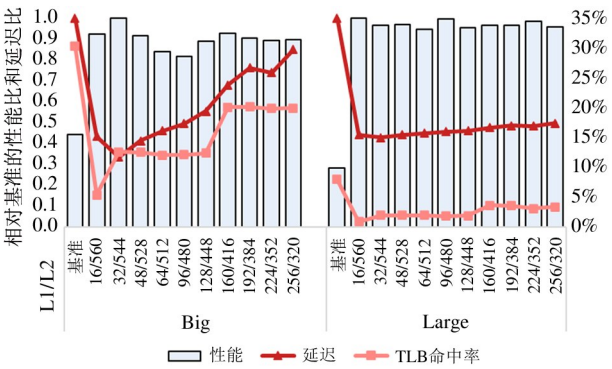

图 11　L1/L2 TLB 容量变化对性能和延迟的影响

PTU 架构设计中,更多的合并项数可以过滤更多的冗余页表遍历请求。本实验探究了合并请求缓冲区中合并项数对性能和延迟的影响。图 12 展示了 4 项(基准)、8 项和 12 项的性能和延迟数据,其中 8 项合并项在 Dense、Big 和 Large 测试集上平均获得了 3.9%、7.8% 和 10.5% 的性能提升,并将每个请求的平均等待延迟降低到了原先的 95.6%、88.0% 和 86.3%。使用 12 项合并项获得的性能提升相比 8 项而言十分有限,仅获得了平均 5.8%、9.9% 和 13.7% 的性能提升。而相比 8 项,12 项合并项却多使用了40.0% 的片上内存资源。因此折

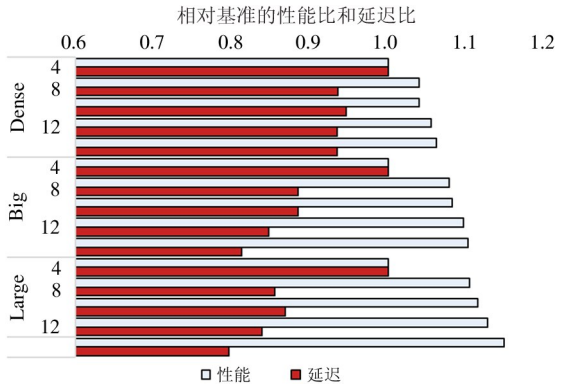

图 12　合并请求缓冲区中合并项对性能和延迟的影响

中考虑,确定使用 8 项合并项的合并请求缓冲区。

本文还探索了 PTU 内多种片上资源的分配方式对性能和延迟的影响。首先,图 13 的实验探究了合并缓冲区项数对性能的影响。实验结果表明,使用更多的合并缓冲区项(16 项以上)在 Dense 和 Big 测试集上并没有明显的性能提升(<2%)。然而,减少合并缓冲区项数会严重影响性能,平均每减少一项合并缓冲区条目,性能就会下降约 5%。这是因为,减少了合并缓冲区项数后,PTU 查找页表的并行数会下降,在面对稀疏访存导致的大量 TLB 丢失的压力下,PTU 并行度下降必然会影响性能。因此,不能减少合并缓冲区条目数。即使是在增加页表缓存的项数的同时,减少合并请求缓冲区条目数(见图 14),MMU 的性能也会大幅下降至 50% ~70%。

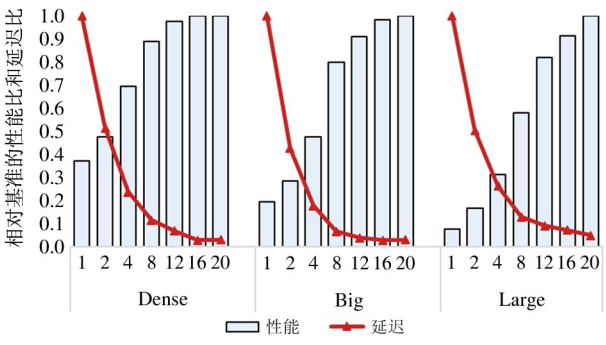

图 13　合并请求缓冲区条目数对性能和延迟的影响

实验发现,适当减少 TPC 的项数对性能和延迟影响不大。当片上资源紧张时,将 32 项的 TPC 减少到 28 项也是可接受的。图 15 展示了实验结果,将 TPC 项数由 32 项调整为 28 项后,Big 和 Large 测试集性能仅损失了约 2.0% 和 8.7%,延迟增加了约

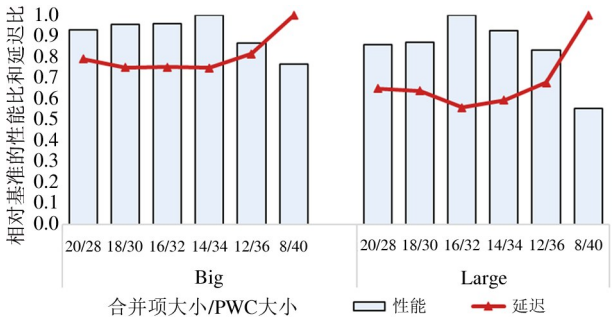

图 14　合并缓冲区和 TPC 大小的变化对性能和延迟的影响

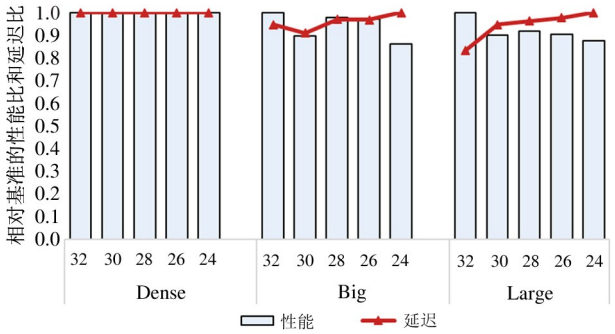

图 15　TPC 项数大小对性能和延迟的影响

2.1% 和 15.0%,但节省了约0.27 kB的片上静态随机存取存储器(static random-access memory,SRAM)资源。Large 测试集的性能损失大于 Big 测试集,这说明更多 TPC 项数可以应对更加稀疏的访存地址。但考虑到 Large 测试集包含了很多极少数情况下才出现的极端稀疏的访存地址,而 Dense 测试集和 Big 测试集的性能和延迟损失都不大,因此减少 TPC 的项数至 28 项是可接受的。

综合 TBU 和 PTU 的 2 处创新设计,HTMMU 在处理 Big 数据集(包含了 DLRM、deepFM 的嵌入表查找和 KGCN 图神经网络聚合操作的部分访存行为)时,其性能相较于基准 MMU 提升了 2.43 倍,相应地请求的平均等待延迟下降为原设计的 34.10%,见图 16。在 45 nm 的工艺下,保守估计 HTMMU 相比原基准 MMU 多使用了 2.75% 的面积。

5　相关工作

系统层面,使用更大的页面大小可以提升 MMU 地址转换的性能。例如现代 x64 系统已支持在不同情形下使用4 kB 和2 MB 页大小。与4 kB 页面相

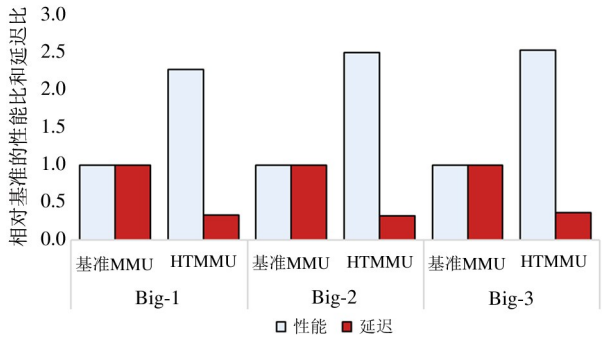

图 16　HTMMU 相比于基准 MMU 在主流算法下性能和延迟优化情况

比,2 MB页面因为覆盖的地址范围更大,局部性更强,因此能大幅提高 TLB 命中率,降低 TLB 缺失导致的延迟。但继续增大页表(比如 1 GB)来获得性能提升是得不偿失的。大页面本身增加了需求分页时数据的传输量,导致大量冗余的数据流量,加重了内存带宽的负担,并且其内部碎片会浪费很多的内存空间。

硬件层面,文献[25]针对各个计算领域中经常出现的基于图的不规则访问请求,提出了图内存管理加速器 Graph MMU,以高效地从片外动态随机存取存储器(dynamic random access memory,DRAM)中获取数据。首先实现了一个基于现场可编程门阵列(field programmable gate array,FPGA)和 ARM CPU 的 AXI(advanced extensible interface)寄存器模式的 AXI DMA 引擎,然后对稀疏的图访存请求序列进行特殊编码,使之成为由本地块随机存取存储器(block random access memory,BRAM)托管的 AXI 描述符链,进而驱动整个 AXI DMA 引擎,以代替 CPU 访问 DRAM,获得了相较于传统 CPU 缓存方法 7 倍吞吐量。然而 GraphMMU 仅针对特殊的图稀疏访存形式(CSR 压缩格式),且某些优化细节和方法与 ARM 体系结构绑定,可扩展性不强。

文献[26]提出了一个新的硬件引擎 SPiDRE,通过近内存数据重组来加速这些应用程序含有稀疏和不规则内存访问模式的应用程序。然而该方法为减少面积开销,直接使用了线性的地址映射模式,这种简单的线性地址映射无法适用于很多复杂场景,比如设备间数据搬运或多级页表等。

算法层面,针对人工智能算法在内存资源受限的设备上部署问题已有大量研究。算法工程师通过使用分解、量化、剪枝、蒸馏等模型压缩技术对算法模型进行压缩,然而很多关于稀疏加速和模型压缩的算法需要与硬件架构设计相互配合,有很强的局限性;很多模型压缩和加速方法或牺牲了算法的精度,或需要大量人工调试。总之,深度学习模型压缩与加速技术尚未发展成熟,在实际部署和产品化水平上还有很大的进步空间[27]。

综上所述,在无法很好地解决资源受限场景下稀疏性和突发性的访存问题时,从现有硬件架构入手,加强包括 MMU 在内的访存通路的性能和吞吐有重要研究意义。

6　结　论

本文深入研究了深度学习算法中不规则的访存行为,旨在缓解片上内存资源严格受限场景下的 MMU 地址转换瓶颈问题。量化分析了当前深度学习算法中存在的不规则访存行为,通过数据分析指出了当前 MMU 架构设计中普遍存在的吞吐率不足的问题。在 gem5 模拟平台上搭建了传统 MMU 架构的地址转换仿真模型。提出了一种高吞吐的 MMU 架构 HTMMU,在传统 MMU 架构的基础上,通过多流并行化 TLB,加强冗余请求过滤,合理分配有限片上存储资源等手段提升了 MMU 的吞吐率,降低了请求平均延迟。

随着深度学习技术的不断发展,在可预见的未来必然有大量人工智能应用会被部署于移动端设备上。本文致力于改进移动端设备上的 MMU,以期其能更适应人工智能应用带来的新变化,使得它在进程间隔离、内存权限保护和芯片间的数据通信等方面发挥更出色的性能,因此具有重要研究意义。

参考文献

[1] NAUMOV M, MUDIGERE D, SHI H J M, et al. Deep learning recommendation model for personalization and recommendation systems [EB/OL]. (2019-03-31) [2023-02-01]. https://arxiv.org/pdf/1906.00091.pdf.

[2] HE X, LIAO L, ZHANG H, et al. Neural collaborative filtering[C]// Proceedings of the 26th International Con-

ference on World Wide Web. Suzhou, China: IEEE, 2017:173-182.

[3] PARK J, NAUMOV M, BASU P, et al. Deep learning inference in facebook data centers: characterization, performance optimizations and hardware implications[EB/OL]. (2018-11-24)[2023-02-01]. https://arxiv.org/pdf/1811.09886.pdf.

[4] GENG T, LI A, SHI R, et al. AWB-GCN: a graph convolutional network accelerator with runtime workload rebalancing[C]// The 53rd Annual IEEE/ACM International Symposium on Microarchitecture. Athens, Greece:IEEE, 2020:922-936.

[5] YAN M, DENG L, HU X, et al. HyGCN: a GCN accelerator with hybrid architecture[C]//2020 IEEE International Symposium on High Performance Computer Architecture (HPCA). San Diego, USA: IEEE, 2020:15-29.

[6] HAN S, POOL J, TRAN J, et al. Learning both weights and connections for efficient neural network[C]// Proceedings of the 28th International Conference on Neural Information Processing Systems. Montreal, Canada: MIT Press, 2015:1135-1143.

[7] YANG Q, MAO J, WANGZ, et al. DASNet: dynamic activation sparsity for neural network efficiency improvement[C]//2019 IEEE 31st International Conference on Tools with Artificial Intelligence. Portland, USA: IEEE, 2019:1401-1405.

[8] LI E, ZENG L, ZHOUZ, et al. Edge AI: on-demand accelerating deep neural network inference via edge computing[J]. IEEE Transactions on Wireless Communications, 2019,19(1):447-457.

[9] HYUN B, KWON Y, CHOI Y, et al. NeuMMU: architectural support for efficient address translations in neural processing units[C]// Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems. Lausanne, Switzerland: ACM, 2020:1109-1124.

[10] ZHANG W, DU T, WANG J. Deep learning over multi-field categorical data: a case study on user response prediction[C]// Proceedings of the 38th European Conference on Advances in Information Retrieval Research. Padua, Italy: Springer International Publishing, 2016: 45-57.

[11] CHENG H T, KOC L, HARMSEN J, et al. Wide & deep learning for recommender systems[C]//Proceedings of the 1st Workshop on Deep Learning for Recommender Systems. Boston, USA: Association for Computing Machinery, 2016:7-10.

[12] WANG Z, WEI Y, LEE M, et al. Merlin hugeCTR: GPU-accelerated recommender system training and inference[C]//Proceedings of the 16th ACM Conference on Recommender Systems. Seattle, USA: Association for Computing Machinery, 2022:534-537.

[13] WANG H, ZHAO M, XIEX, et al. Knowledge graph convolutional networks for recommender systems[C]// The World Wide Web Conference. San Francisco, USA: Association for Computing Machinery, 2019:3307-3313.

[14] WANG Z, WANG Y, YUAN C, et al. Empirical analysis of performance bottlenecks in graph neural network training and inference with GPUs[J]. Neurocomputing, 2021,446:165-191.

[15] KIPF T N, WELLING M. Semi-supervised classification with graph convolutional networks[EB/OL]. (2016-09-09)[2023-02-01]. https://arxiv.org/pdf/1609.02907.pdf.

[16] LI Y, TARLOW D, BROCKSCHMIDT M, et al. Gated graph sequence neural networks[EB/OL]. (2015-11-17)[2023-02-01]. https://arxiv.org/pdf/1511.05493.pdf.

[17] HENNESSY J L, PATTERSON D A. Computer architecture: a quantitative approach[M]. Amsterdam: Elsevier, 2011.

[18] COX G, BHATTACHARJEE A. Efficient address translation for architectures with multiple page sizes[J]. ACM SIGPLAN Notices, 2017,52(4):435-448.

[19] BARR T W, COX A L, RIXNER S. Translation caching: skip, don't walk (the page table)[J]. ACM SIGARCH Computer Architecture News, 2010,38(3):48-59.

[20] ANATI I, BLYTHE D, DOWECK J, et al. Inside 6th gen Intel ® Core™: new microarchitecture code named skylake[C]//2016 IEEE Hot Chips 28 Symposium. Cupertino, USA: IEEE, 2016:1-39.

[21] GUO H, TANG R, YE Y, et al. DeepFM: a factorization-machine based neural network for CTR prediction[EB/OL]. (2017-03-13)[2023-02-01]. https://arxiv.org/pdf/1703.04247.pdf.

[22] BHATTACHARJEE A. Large-reach memory management

unit caches[C]//Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture. Davis, USA: IEEE, 2013:383-394.

[23] BINKERT N, BECKMANN B, BLACK G, et al. The gem5 simulator[J]. ACM SIGARCH Computer Architecture News, 2011,39(2):1-7.

[24] LOWE-POWER J, AHMAD A M, AKRAM A, et al. The gem5 simulator: version 20.0 +[EB/OL]. (2020-07-07)[2023-02-01]. https://arxiv.org/pdf/2007.03152.pdf.

[25] KAPRE N, HAN J L, BEAN A, et al. GraphMMU: memory management unit for sparse graph accelerators[C]//2015 IEEE International Parallel and Distributed Processing Symposium Workshop. Hyderabad, India: IEEE, 2015:113-120.

[26] BARREDO A, BEARD J C, MORETÓ M. Poster: spidre: accelerating sparse memory access patterns[C]//2019 28th International Conference on Parallel Architectures and Compilation Techniques. Seattle, USA: IEEE, 2019:483-484.

[27] 高晗, 田育龙, 许封元, 等. 深度学习模型压缩与加速综述[J]. 软件学报, 2020,32(1):68-92.

HTMMU: a memory management unit for irregular memory access in deep learning

DING Feng, LI Xi

(School of Computer Science and Technology, University of Science and Technology of China, Hefei 230026)

Abstract

The diversification and complexity of artificial intelligence applications lead to irregular memory access pattern. The irregular memory access pattern can be defined as bursty and sparse memory access requests, which brings great challenges to the deployment of intelligent applications on mobile devices with strictly limited memory resources. This irregular memory access pattern has caused the memory management unit (MMU) in existing architectures to face the problems of low throughput and long latency, making it a bottleneck of the system. To solve this problem, this paper proposes a novel MMU architecture called high-throughput MMU (HTMMU). HTMMU uses multi-stream parallelism, enhances filtering of redundant requests and allocates limited on-chip memory more reasonably to improve system memory access efficiency. Experimental results show that when dealing with the irregular memory accesses in artificial intelligence algorithms, compared with the current MMU design, HTMMU achieves 2.43 times speedup averagely, and reduces the average latency by 65.9% with less than 3.0% area overhead.

Key words: memory management unit (MMU), address translation, irregular memory access, deep learning, high-throughput