软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2023,34(12):5862−5886 [doi: 10.13328/j.cnki.jos.006800]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

面向深度学习训练的内存交换机制综述*

高赫然[1,2], 吴 恒[2,5,6], 许源佳[1,2], 李修和[4], 王 焘[2,3], 张文博[2,3,5,6]

[1](中国科学院大学, 北京 100049)

[2](中国科学院 软件研究所 软件工程技术研究开发中心, 北京 100190)

[3](计算机科学国家重点实验室 (中国科学院 软件研究所), 北京 100190)

[4](国防科技大学 电子对抗学院, 安徽 合肥 230037)

[5](中国科学院大学南京学院, 江苏 南京 211135)

[6](中科南京软件技术研究院, 江苏 南京 210000)

通信作者: 张文博, E-mail: zhangwenbo@otcaix.iscas.ac.cn

摘 要: 随着深度学习技术的快速发展和深入应用, 深度学习训练规模持续增大, 内存不足已成为影响深度学习可用性的主要瓶颈之一. 内存交换机制是应对深度学习训练内存问题的关键技术, 该机制利用深度学习训练内存需求的“时变”特征, 在专用计算加速设备内存与外部存储之间按需移动数据, 通过瞬时内存需求替代累积内存需求, 保障深度学习训练任务的运行. 对面向深度学习训练的内存交换机制进行综述, 以深度学习训练内存需求的时变特征为研究视角, 分别针对基于算子运行特征的内存换出机制、基于数据依赖关系的内存换入机制以及效能驱动的联合换出与换入决策等重要研究工作进行了总结分析, 并针对该技术领域的发展方向进行了展望.

关键词: 深度学习训练; 内存交换; 内存需求特征

中图法分类号: TP303

中文引用格式: 高赫然, 吴恒, 许源佳, 李修和, 王焘, 张文博. 面向深度学习训练的内存交换机制综述. 软件学报, 2023, 34(12): 5862–5886. http://www.jos.org.cn/1000-9825/6800.htm

英文引用格式: Gao HR, Wu H, Xu YJ, Li XH, Wang T, Zhang WB. Survey on Memory Swapping Mechanism for Deep Learning Training. Ruan Jian Xue Bao/Journal of Software, 2023, 34(12): 5862–5886 (in Chinese). http://www.jos.org.cn/1000-9825/6800.htm

Survey on Memory Swapping Mechanism for Deep Learning Training

GAO He-Ran[1,2], WU Heng[2,5,6], XU Yuan-Jia[1,2], LI Xiu-He[4], WANG Tao[2,3], ZHANG Wen-Bo[2,3,5,6]

[1](University of Chinese Academy of Sciences, Beijing 100049, China)

[2](Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

[3](State Key Laboratory of Computer Science (Institute of Software, Chinese Academy of Sciences), Beijing 100190, China)

[4](Institute of Electronic Countermeasures, National University of Defense Technology, Hefei 230037, China)

[5](University of Chinese Academy of Sciences, Nanjing, Nanjing 211135, China)

[6](Nanjing Institute of Software Technology, Nanjing 210000, China)

Abstract: With the rapid growth and further application of deep learning (DL), the scale of DL training continues to expand, and memory insufficiency has become one of the major bottlenecks threatening DL availability. Memory swapping mechanism is the key mechanism to alleviate the memory problem of DL training. This mechanism leverages the “time-varying” memory requirement of DL training and

* 基金项目: 国家重点研发计划 (2018YFB1402803); 国家自然科学基金 (61872344, 61972386); 山东省重大研发计划 (2021CXGC010101)
高赫然, 吴恒为共同第一作者.
收稿时间: 2022-04-27; 修改时间: 2022-06-12, 2022-08-09; 采用时间: 2022-09-17; jos 在线出版时间: 2022-12-20
CNKI 网络首发时间: 2023-04-27

moves the data between specific computing accelerating device memory and external storage according to demands. The operation of DL training tasks can be ensured by replacing an accumulated memory requirement with an instant one. This study surveys the memory swapping mechanism for DL training from the aspect of time-varying memory requirements. Key studies of an operator feature-based memory swapping-out mechanism, a data dependency based swapping-in mechanism, and efficiency-driven joint swapping-in and swapping-out decisions are summarized. Finally, the development prospect of this technology is pointed out.

Key words: deep learning (DL) training; memory swapping; memory requirement characteristic

近年来, 以深度学习为代表的新型计算范式取得了长足进步, 并在计算机视觉[1−3]、自然语言处理[4−6]等领域广泛应用. 但随着深度学习模型大小和数据集规模持续增加[7], 内存不足问题日益突出. 例如, BERT 模型的内存使用可达到 175 GB, 而 GPT-3 模型具有 1 700 亿参数[8], 内存使用可达到 TB 级别. 内存管理问题已成为影响深度学习可用性与性能的重大挑战.

为解决高内存需求下深度学习训练的可用性问题, 现有工作主要采用分布式训练与模型压缩的方案. 分布式训练[9,10]本质是扩充内存容量, 通过数据并行与模型并行, 将深度学习训练扩展到多个专用计算加速设备. 如 GShard 使用了 2 048 个 TPU 进行分布式训练, 总内存使用达到 2.4 TB[11]; 模型压缩[12,13]本质是减少内存需求, 通过网络剪枝、网络精馏、网络分解等方法, 牺牲一定的模型准确性, 满足模型在专用计算加速设备上的运行[14]. 然而, 这些方法的有效性以准确的内存容量规划为基础, 实用性较差.

内存交换机制是近年来应对内存管理挑战的一种重要技术, 其核心思想是利用深度学习训练内存需求的“时变”特征, 通过在某一时间点将内存数据从专用计算加速设备内存中动态移出, 并在数据被访问时从外部存储将其重新移回专用计算加速设备内存, 以瞬时内存需求替代累积内存需求, 达到深度学习训练累积内存可以超出单个专用计算加速设备的内存容量的效果, 保障深度学习训练在此设备上的可用性与性能.

目前, 现有综述工作主要关注分布式训练[9,10]与模型压缩机制[12−14], 尚未对内存交换机制进行完整全面地综述. 马玮良[15]在对内存管理机制进行的综述中介绍了包含内存交换的一系列内存管理方案及相关代表性工作, 但总体而言, 现有综述工作对于以内存需求特征为视角的内存交换机制仍然缺乏关注. 本文着重关注于内存交换机制, 以深度学习训练内存需求的时变特征为视角, 对基于算子运行特征的内存换出机制、基于数据依赖关系的内存换入机制以及效能驱动的联合换出与换入决策等本领域重要研究工作进行系统地总结和分析, 并对相关技术的发展方向进行展望.

本文第 1 节介绍内存交换的目标及其面临的挑战, 并引出本文研究框架. 第 2 节介绍基于算子运行特征的内存换出机制. 第 3 节介绍基于数据依赖关系的内存换入机制. 第 4 节介绍效用驱动的内存换出与换入联合决策. 第 5 节介绍内存交换机制的应用. 第 6 节对未来的研究方向进行展望. 第 7 节总结全文.

1 研究问题概述

1.1 研究目标

内存交换机制通过在专用计算加速设备内存和外部存储之间换出换入内存数据, 以瞬时内存需求替代累积内存需求作为内存供给依据, 保障深度学习训练的运行. 具体而言, TensorFlow[16]、PyTorch[17]等主流深度学习框架将深度学习模型建构为算子计算图 $G=\{V,E\}$, 其中算子集合 $V=\{v_1,v_2,\ldots,v_n\}$ 表示卷积、矩阵乘法等计算操作, $E=\{e_1,e_2,\ldots,e_m\}$, $e=<v_i,v_j>$, $v_i\in V$, $v_j\in V$ 表示算子间输入输出关系. 则内存交换机制在深度学习训练的每一时刻 i 构建 $V_{\text{dev}}^{(i)}\subseteq V$ 与 $V_{\text{storage}}^{(i)}\subseteq V$, 分别表示此时刻位于专用计算加速设备与外部存储设备的算子, 构建 $V_{\text{SO}}^{(i)}\subseteq V$ 与 $V_{\text{SI}}^{(i)}\subseteq V$, 分别表示此时刻换出与换入数据的算子, 其中, $V_{\text{dev}}^{(i)}\cap V_{\text{storage}}^{(i)}=\varnothing$, $V_{\text{SO}}^{(i)}\cap V_{\text{SI}}^{(i)}=\varnothing$. 在应用内存交换机制时, 每一时刻 i 的 $V_{\text{SO}}^{(i)}$ 与 $V_{\text{SI}}^{(i)}$ 可通过如下方式计算:

$$V_{\text{dev}}^{(i)}=\begin{cases}\{v_1\}, & i=1\\ (V_{\text{dev}}^{(i-1)}\cup V_{\text{allocated}}^{(i)}\cup V_{\text{SI}}^{(i)})\backslash(V_{\text{freed}}^{(i)}\cup V_{\text{SO}}^{(i)}), & i\geqslant 2\end{cases} \tag{1}$$

$$V_{\text{storage}}^{(i)}=\begin{cases}\varnothing, & i=1\\ (V_{\text{storage}}^{(i-1)}\cup V_{\text{SO}}^{(i)})\backslash V_{\text{SI}}^{(i)}, & i\geqslant 2\end{cases} \tag{2}$$

其中, $V_{\text{allocated}}^{(i)}\subseteq V$ 为此时刻分配内存空间并开始运行的算子, $V_{\text{freed}}^{(i)}\subseteq V$ 为此时刻结束运行并释放内存空间的算子. 内存交换机制需要为深度学习训练的每一时刻 i 构建 $V_{\text{SO}}^{(i)}$ 与 $V_{\text{SI}}^{(i)}$, 并实现以下两方面具体目标.

(1) 可用性目标: 降低深度学习训练的瞬时内存需求. 内存交换机制在深度学习训练的每一时刻 i 构建 $V_{\text{SO}}^{(i)}$, 选择算子数据换出内存, 实现:

$$f_{\text{mem}}(V_{\text{dev}}^{(i)})<m_{\text{dev}} \tag{3}$$

其中, f_{mem} 为计算算子集合 $V_{\text{dev}}^{(i)}$ 内存占用的函数, m_{dev} 为专用计算加速设备的内存容量. 内存交换机制通过选择算子数据换出内存, 保障深度学习瞬时内存需求低于专用计算加速设备的内存容量, 不因内存不足而崩溃.

(2) 性能目标: 减少深度学习训练阻塞. 内存交换机制定义深度学习训练每一时刻 i 所需算子的集合. $V_{\text{required}}^{(i)}\subseteq V$, 并构建 $V_{\text{SI}}^{(i)}$, 选择算子数据换入内存, 实现:

$$V_{\text{required}}^{(i)}\cap V_{\text{storage}}^{(i)}=\varnothing \tag{4}$$

$$\forall v\in V_{\text{required}}(<v',v>\in E\rightarrow v'\in V_{\text{required}}) \tag{5}$$

其中, 运行中每一时刻 i 所需的算子属于 $V_{\text{required}}^{(i)}$. 而 $V_{\text{required}}^{(i)}$ 中算子的数据需要处于专用计算设备的内存之中, 保障深度学习训练不因等待内存交换而阻塞. 内存交换机制通过为每一算子确定交换内存数据的时刻, 减少深度学习训练因内存交换产生的阻塞, 保障深度学习训练性能.

1.2 挑　战

为了实现可用性与性能目标, 内存交换机制需要选择内存不足时换出的算子数据, 确定其换入内存的时刻, 最终产生可行的内存交换方案. 在这一过程中, 内存交换机制具体面临如下 3 方面的挑战.

(1) 如何确定从内存换出的算子数据. 内存交换机制需要在专用计算加速设备内存不足时选择算子数据换出内存. 若没有充分考虑内存需求的时变特征, 将可能无法确定最适合换出内存的算子数据, 例如换出即将被访问的算子数据, 导致难以有效削减深度学习训练的瞬时内存需求, 并可能因满足可用性目标进行大量内存交换, 显著影响训练性能. 因此, 内存交换机制需要结合算子运行特征构建换出机制, 确定从内存换出的算子数据, 满足可用性目标.

(2) 如何确定算子数据换入到内存的时刻. 内存交换机制需要在换出的数据被访问时, 将其及时换入内存. 若没有充分考虑内存需求的时变特征, 将可能无法在最准确的时刻将算子数据换入内存, 导致深度学习训练阻塞等待内存数据换入完成, 显著影响训练性能. 因此, 内存交换机制需要结合数据依赖关系, 构建换入机制, 确定将算子数据换入到内存的时刻, 满足性能目标.

(3) 如何避免算子换出与换入冲突. 内存交换机制需要通过内存换出与换入的联合决策, 形成训练运行阶段的内存交换方案. 若单独决策内存换出与换入, 而没有充分考虑深度学习训练运行与内存交换共同决定的时变特征, 将可能导致换出与换入决策间存在冲突, 难以保障训练性能. 因此, 内存交换机制需要结合考虑深度学习训练运行状况和内存交换优化策略等运行阶段因素, 进行效用驱动的联合决策, 避免算子换出与换入决策冲突, 以最终满足可用性与性能目标.

1.3 研究框架

本文提出如图 1 所示的研究框架. 为了兼顾可用性与性能目标, 保障深度学习训练的运行, 内存交换机制需要有效结合内存需求时变特征, 解决换出机制、换入机制与联合决策中所面临的一系列挑战.

基于算子运行特征的内存换出机制解决算子数据的换出选择挑战. 内存换出机制在内存换出阶段结合体现为算子运行特征的时变特征, 选择算子数据换出内存, 以满足可用性目标.

基于数据依赖关系的内存换入机制解决算子数据的换入时刻挑战. 内存换入机制需要在内存换入阶段结合体

现为换出数据间的依赖关系的时变特征, 确定数据访问时刻, 通过反馈或前馈方式及时将算子数据从外部存储换入内存, 以满足性能目标.

效用驱动的内存交换联合决策解决算子数据交换的联合决策挑战. 内存交换机制结合换出与换入机制, 考虑体现时变特征的深度学习训练因素与影响时变特征的内存交换策略因素, 通过训练效能驱动的联合交换决策, 兼顾可用性与性能目标, 保障深度学习训练的运行.

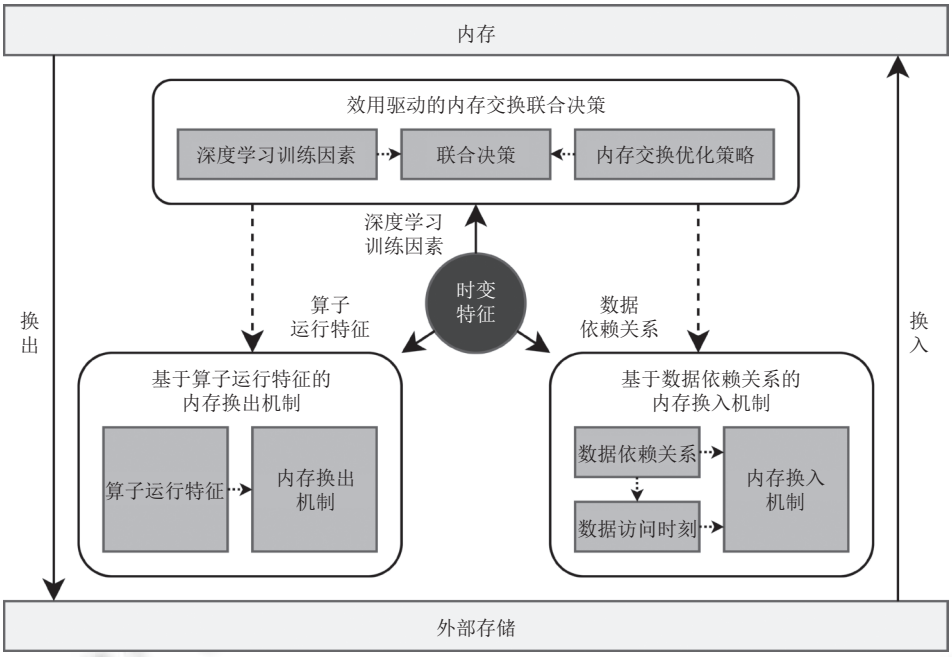


图 1 研究框架

2 基于算子运行特征的内存换出机制

当深度学习训练的内存需求超出于专用计算加速设备的内存供给时, 深度学习框架需要结合训练阶段的内存需求时变特征构建内存换出机制, 选择算子数据换出内存, 为深度学习训练释放内存资源, 有效实现内存交换的可用性目标.

第 2.1 节介绍深度学习算子的运行特征, 第 2.2 节介绍深度学习框架面向这些特征提供的内存换出机制, 第 2.3 节对本节进行总结.

2.1 算子运行特征

2.1.1 内存先入后出特征

深度学习训练中最先计算的算子的数据最先进入内存, 最后从内存中释放, 具有最长的内存占用时间. 图 2(a) 展示了算子数据在前向传播与反向传播中的复用. 在 Op2 的前向传播中, X、Y 分别为算子的输入与输出数据, W 为算子权重数据, 其中, Y 由 X 与 W 通过算子的前向传播操作 op 计算产生; 在 Op2 反向传播 (Op2′) 中, dX 与 dY 分别为算子反向传播中的梯度数据, 其中, dX 由 X、Y 与 dY 通过算子的反向传播操作 dop 计算产生. 可以发现, 算子在深度学习训练前向传播中的输出数据在由下一算子使用之外, 还需要在反向传播中复用, 需要保留在内存之中. 图 2(b) 展示了这一训练机制对于内存占用时间的影响. 前向传播与反向传播按相反顺序进行, 因此, 最先前向传播的算子最后进行反向传播, 最先被分配, 而最后被释放内存空间, 因此, 深度学习框架可以将这些算子的内存数据换出内存, 释放内存空间, 降低深度学习训练的瞬时内存需求[18−21].

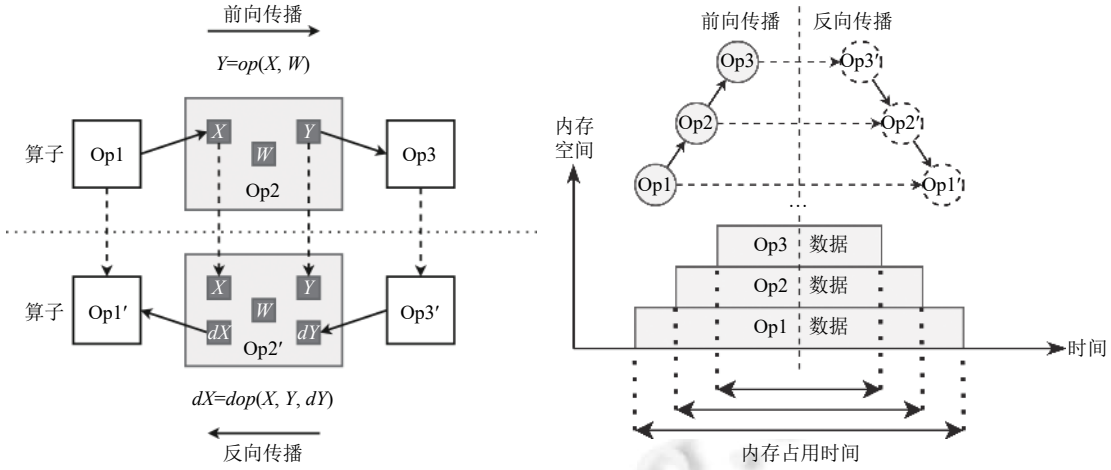


图 2 内存先入后出特征

2.1.2 算子内存占用差异特征

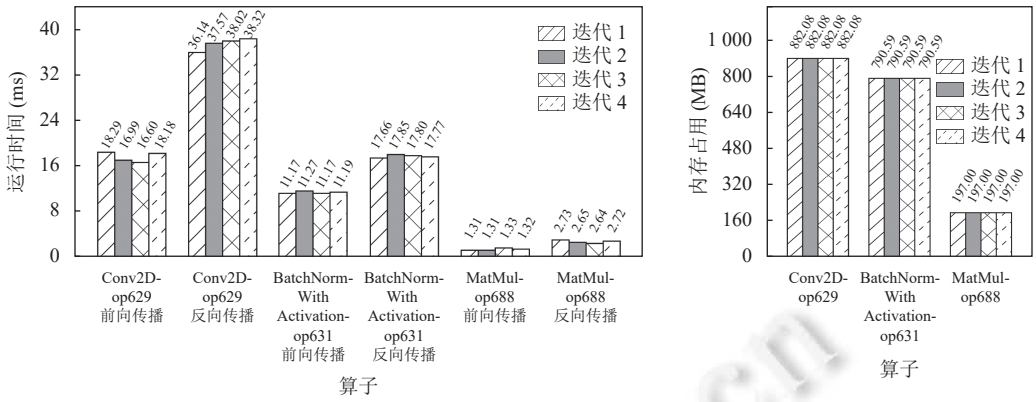
由于在深度学习模型中位于不同位置, 实现不同功能, 算子之间存在较大的内存占用差异. 表 1 展示了使用华为 MindSpore 深度学习框架训练 VGG-16模型时部分算子权重、输出与工作区域的内存占用. 不同类别的算子具有显著的内存占用差异, 卷积算子与用于实现全连接层的矩阵乘法算子相比, 占用了更多的内存空间. 而且随着对于输入数据特征提取的不断深入, 卷积算子的内存占用逐渐减少, 例如 Conv2D-op651 与 Conv2D-op629 相比, 具有减半的内存占用. 现有工作发现并总结这一特征, 对于 CNN 模型, 卷积算子具有较高的内存占用[18]; 对于 RNN 模型, 实现 attention 机制的加法算子具有较短的计算时间, 但其中间结果需要占用大量内存空间[20,22]. 内存交换时, 深度学习框架换出更大的算子可以释放更多的内存空间. 不同算子的不同组成部分也存在较大的内存占用差异, 卷积算子中输入输出数据具有较大的比重, 例如表 1 展示的卷积算子中, 输出数据占用了绝大部分的内存空间, 对于 Conv2D-op651 算子, 其权重与工作区域的总和小于 1 MB; 而实现全连接层的矩阵乘法算子中权重具有较大的比重[18]. 矩阵乘法算子输出大小为 1 MB, 而权重输入可以达到数十至数百 MB. 深度学习框架选择交换 CNN 模型卷积算子的输入输出数据[18,23,24], 或 RNN 模型的 attention 中间结果[20]. 部分工作也提出调整卷积等算子的实现, 消除算子工作区的内存占用[18].

表 1 VGG-16 模型算子内存需求

算子	权重/输入大小 (MB)	输出大小 (MB)	工作区大小 (MB)
Conv2D-op629	0.07	784.00	98.01
BatchNormWithActivation-op631	0.00	784.00	6.59
Conv2D-op651	0.28	392.00	0.35
BatchNormWithActivation-op653	0.00	392.00	3.29
MatMul-op688	196.00 (输入数据)	1.00	0.00
MatMul-op691	32.00 (输入数据)	1.00	0.00

2.1.3 迭代间稳定性特征

图 3 展示了 MindSpore 框架中部分算子在迭代间的运行状况. 可以发现每次迭代中, 算子在前向传播与反向传播阶段具有稳定的运行时间与内存需求. 例如, 卷积算子的前向传播时间稳定在 16–18 ms, 反向传播时间稳定在 36–38 ms. 其原因在于每次迭代中只有输入数据内容发生变化, 而影响输入数据规模的批处理尺寸、影响计算开销的内部实现等参数配置没有发生变化. 深度学习框架对这一特征的利用分为两方面. 当深度学习训练的参数与资源配置不变时, 深度学习框架使用这一特征通过获取一次迭代中的算子运行状况, 用于后续数十万次迭代中内存交换的决策[18,25]. 而训练参数与资源配置改变时, 深度学习框架可以快速评估算子运行状况的变化, 针对性调整内存换出机制[26–28].



(a) 算子运行时间 (b) 算子内存占用

图 3 迭代间稳定性特征

2.2 内存换出机制

2.2.1 先出后入换出机制

深度学习框架基于算子内存先入后出的特征, 采用先出后入的换出机制. 深度学习框架根据其对于训练阶段可用性与性能的要求, 采用即时换出或按需换出的方式.

(1) 即时换出. 深度学习框架在前向传播某一算子数据不再被使用时, 换出其内存数据. Capuchin[19]在某一算子内存数据不再使用时, 即时将其换出内存, Meng 等人[20]选择换出最先进行前向传播的算子数据. FlashNeuron[29]基于算子前向与反向计算时间决定是否换出某一算子的数据, 并根据训练的运行性能调整选择的算子. HALO[30]基于算子数据量构建内存换出机制, 并指出具有较大数据量的算子通常在模型的前半部分, HALO 在运行时结合算子数据量、热度、生命周期等参数计算数据的交换价值, 通过装箱算法即时换出算子数据.

(2) 按需换出. 深度学习框架在内存不足时, 通过内存换出机制释放内存空间. moDNN[21]、SuperNeurons[23]、Tahoe[31]使用 LRU 算法换出最近最少使用的算子数据, 而最近最少使用的算子通常是最先进行前向传播的算子. 深度学习框架通常结合由算子间依赖关系决定的内存访问模式, Layrub[24]、Dymem[32]使用引用计数描述运行阶段的依赖状况, 在前向传播算子先出后入之上, 当算子的引用计数减为 0 时移出其数据, 避免换出即将被访问的内存数据.

2.2.2 大算子优先换出机制

深度学习框架基于算子数据对内存资源的占用空间差异, 在内存数据换出时, 通过卸载占用大量内存的算子数据, 有效降低算子的内存占用. 深度学习框架根据从深度学习训练过程中观察总结的算子内存占用差异特征, 或在运行时动态选择换出算子数据.

(1) 观察总结的内存占用状况. vDNN[18]、SuperNeurons[23]、Layrub[24]、HALO[30]等工作面向 CNN 模型, 观察到卷积等算子具有较高的内存占用, vDNN、Layrub 还观察到卷积等算子的内存占用主要集中在输入输出数据, 并选择其数据进行交换. Meng 等人[20]的工作与 Echo[22]面向 RNN 模型, 观察到实现注意力机制的加法算子需要保留大量的中间输入输出数据, 并将这些数据换出内存. 在这一机制下, 深度学习框架在选择换出的算子数据时, 仅考虑算子数据的类别, 而不考虑算子数据的实际数据量.

(2) 动态获取的内存占用状况. Zhang 等人[33]的工作在训练运行阶段根据算子的内存占用情况, 选择算子数据换出内存. Capuchin[19]根据算子数据量计算换出算子的收益, 选择具有最大内存释放效果的算子. FlashNeuron[29]首先换出所有算子, 随后根据算子数据量决定是否将其保留在内存中. 算子因处于模型的不同位置, 其数据量可能存在较大的差异, 深度学习框架可以结合观察总结的算子内存占用差异, 限制动态选择换出算子的范围, 在卷积等算

子中选择占用更多内存空间的算子, 降低动态选择的开销.

2.2.3 周期性换出机制

深度学习框架基于算子迭代间的稳定性特征, 在每次迭代中收集运行状况信息, 周期性换出内存数据. vDNN[18]、FlashNeuron[29]等工作在深度学习训练启动时进行通过试运行, 生成内存换出方案. PoocH[25]使用若干组参数进行训练, 根据收集到的性能与内存使用数据构建预测模型, 用于构建内存换出方案. Capuchin[19]在训练的第 1 个迭代中收集内存访问信息, 用于后续的内存换出与换入联合决策. Meng 等人[20]的工作, 以及 LMS[34], 将换出决策写入到算子计算图中, 以在训练后续迭代的运行中自动执行换出决策.

2.2.4 内存换出机制的算法描述

算法 1 展示了内存换出机制. 深度学习框架在内存不足时按需换出数据, 并在算子运行结束之后进行即时换出, 为深度学习训练持续供给内存. 深度学习框架结合多种内存换出机制, 选择算子数据换出内存, 保障深度学习训练的运行.

算法 1. 内存换出机制.

输入: 深度学习训练任务 f, 时刻 i;

输出: 无.

1. $V_{\text{dev}}^{(i)}=f.G.V_{\text{dev}}^{(i)}$;
2. $i=0$;
3. $m_{\text{required}}=fm(V_{\text{dev}}^{(i)})-m_{\text{dev}}$;
4. while $m_{\text{required}}>0$ do //累积内存需求超出于专用计算设备内存容量, 需要应用内存换出机制, 进行按需换出
5. v_{swapout} = selectOperatorOnDevice(); //使用 LRU 等算法选择算子, 或应用大算子优先换出机制
6. if $\exists v' \in V_{\text{required}}(<v_{\text{swapout}}, v'>\in E)$ then //换出的算子受到依赖
7. continue;
8. end
9. $V^{(i)}_{\text{SO}}=V^{(i)}_{\text{SO}} \cup \{v_{\text{swapout}}\}$;
10. $m_{\text{required}}=m_{\text{required}}-v_{\text{swapout}}.\text{size}()$;
11. end
12. swapout($V_{\text{SO}}^{(i)}$); //执行换出
13. f.execute(); //开始运行
14. for v_{required} in V_{required} //算子运行结束, 进行即时换出
15. if $\exists v' \in V(<v_{\text{required}}, v'>\in E)$ then //算子后续受依赖, 暂时不应换出
16. continue;
17. end
18. $V_{\text{SO}}^{(i)}=V_{\text{SO}}^{(i)} \cup \{v_{\text{required}}\}$;
19. end

2.3 对比与小结

表 2 总结了部分工作的换出决策. 深度学习框架的内存换出机制通过有效结合体现为深度学习算子运行特征的内存时变需求特征, 恰当选择算子内存数据换出内存, 有效降低训练阶段的瞬时内存需求. 深度学习框架通过结合多种算子运行特征, 构建多种内存交换机制, 最大化降低训练阶段的瞬时内存需求, 为深度学习训练释放内存空间. 进一步探索内存使用特征及其与换出机制结合的方法, 将可能进一步降低深度学习训练瞬时内存需求, 满足内存交换的可用性与性能目标.

表 2 基于算子运行特征的内存换出机制

成果	交换粒度	内存类别	内存使用特征	换出机制	特征应用方式	效果
Dymem[32]	算子数据	GPU内存	内存先入后出	先出后入	确定算子引用计数, 触发内存换出	在vDNN[18]之上降低31% 内存占用
Echo[22]	算子数据	GPU内存	不同类别算子内存占用差异	大算子优先	重计算大规模、短计算时间的算子	在MXNet之上减少内存占用(平均1.89倍, 最大3.13倍)
Layrub[24]	算子数据	GPU内存	内存先入后出	先出后入	确定算子引用计数, 触发内存换出	在Caffe之上减少内存占用(平均58.2%, 最大98.9%)
Soft2LM[35]	内存页面	CPU DRAM/NVM	稳定内存使用	周期性交换	根据算子数据对于特定内存类别的亲和性, 确定深度学习训练的内存放置与迁移方案	在PARSEC评测基准上提升应用性能4.7倍
SuperNeurons[23]	卷积算子数据	GPU内存	内存先入后出、算子不同组成部分内存占用差异	先出后入、大算子优先	通过内存最近最少使用刻画算子内存先入后出特征 确定优先交换卷积算子的输入输出数据	在TensorFlow之上支持3.243 2倍更深的神经网络
vDNN[18]	算子数据	GPU内存	稳定内存使用、不同类别算子不同部分的内存使用差异	周期性交换、大算子优先交换	通过试运行生成内存交换方案 确定优先进行交换卷积算子数据	与Torch相比, 减少约90%内存占用(AlexNet 89%, OverFeat 91%, GoogLeNet 95%)
文献[20]	算子输入输出数据	GPU内存	内存先入后出、不同类别算子不同部分内存占用差异	先出后入、大算子优先	确定算子引用计数, 触发内存换出 重计算大规模、短计算时间的算子	在TensorFlow之上减少内存占用(54.26%), 提高批尺寸2到30倍
文献[33]	算子数据	GPU内存	内存先入后出、不同类别算子内存占用差异	先出后入、大算子优先	综合考虑算子数据的内存占用时间与占用空间, 决策内存交换	在cuDNN上降低34.2%的内存占用
文献[36]	算子数据	GPU内存	内存先入后出	先出后入	综合考虑算子热度与访问距离, 决策内存交换	与统一内存基础方法相比, 提升性能7.6倍, 最大18.7倍
文献[37]	内存页面	CPU DRAM/NVM	稳定内存使用	周期性交换	根据算子内存数据对于特定内存类别的亲和性, 确定训练的内存放置与迁移方案	在PARSEC、SPLASH、BigDataBench等评测基准上降低30%的运行时间

3 基于数据依赖关系的内存换入机制

对于已经换出内存的算子数据, 深度学习框架需要利用训练阶段的内存需求时变特征, 为每个换出的算子确定时刻, 将算子数据换入内存, 保障深度学习训练的内存访问. 深度学习训练的内存需求时变特征决定了算子数据被访问的时刻, 而算子运行时间、内存交换时间等因素决定了算子数据被换入的时刻. 深度学习框架需要结合数据依赖关系, 构建反馈与前馈内存数据换入机制, 在特定时刻将算子数据换入内存, 提高内存使用效率, 保障深度学习训练的运行性能目标.

第 3.1 节介绍数据依赖关系的获取方法, 第 3.2 节介绍数据访问时间的获取方法, 第 3.3 节介绍深度学习框架结合数据依赖关系的内存数据换入机制, 第 3.4 节对本节进行总结.

3.1 数据依赖关系的获取方法

深度学习框架需要在访问依赖数据之前将其换入内存. 如图 4 所示, 深度学习框架在编译与运行阶段, 通过计算图分析与监测方法, 提取外部存储中数据的依赖关系.

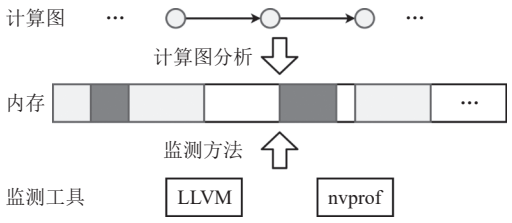


图 4 数据依赖关系的获取方法

3.1.1 计算图分析方法

如图 4 所示, 深度学习框架从训练阶段提取信息, 将算子间依赖关系, 映射到内存数据之间的依赖关系. SuperNeurons[23]为每个算子维护输入输出算子集合, 并在运行时动态调整这些集合, 获取算子间依赖关系. Layrub[24]为算子进行引用计数, 从算子的访问时间序列中获取算子依赖关系. HALO[30]分析深度学习训练的计算图, 分析算子热度与生命周期, 获取算子依赖关系. 算子间依赖关系可以映射到运行时内存数据之上, 用于确定换出的内存数据间的依赖关系.

由于以计算图作为提取数据依赖关系的来源, 深度学习框架获取数据依赖关系的精确度与深度学习框架将计算图节点映射到运行时内存数据的精确度密切相关. 深度学习框架可能进行针对于计算设备的训练阶段优化, 例如针对不同类型的设备编译生成不同的代码, 这对于深度学习框架将计算图节点精确映射到运行时内存数据的能力提出了更高的要求.

3.1.2 监测方法

如图 4 所示, 深度学习框架直接监测训练阶段的内存访问状况, 从底层的内存地址访问记录中获取数据间依赖关系. Li 等人[36]的工作通过 LLVM 编译器的中间表示 (IR) 获取各内存数据的访问信息与重用距离. ETC[37]检查深度学习程序中的指针, 通过内存地址聚合获取内存数据对象的依赖关系. Tahoe[31]使用计算设备监测工具, 获取底层内存地址层级上的内存访问模式, 并结合深度学习训练的语义, 将内存地址访问映射到内存数据依赖关系. Leap[38]设置时间窗口监测内存地址访问, 动态调整窗口大小以提高地址监测的准确性, 并通过 Boyer-Moore 算法将内存地址的访问映射到内存区域的访问. Capuchin[19]在运行时采集算子的数据依赖关系, 若某一算子在运行时访问了另一算子的数据, 则此算子对另一算子存在依赖.

由于在深度学习训练的运行阶段提取信息, 监测方法可以更为准确地获取数据依赖关系. 由于换出的算子数据可能包含大量的内存数据对象, 深度学习框架通过结合上层训练阶段的信息, 将内存地址间依赖关系映射为内存数据对象间依赖关系. 数据依赖关系的获取效果取决于深度学习框架结合与利用训练阶段信息的能力.

3.2 数据访问时间的获取方法

深度学习框架需要将数据依赖关系与数据访问时间结合, 确定数据在何时需要换入内存. 数据访问时间基于算子运行时间与内存交换时间计算, 用于对齐算子运行时间与内存交换时间, 实现在深度学习训练访问数据的同一时刻完成数据换入. 根据深度学习训练配置的稳定性, 深度学习框架采用监测与预测的方法获取运行时间信息.

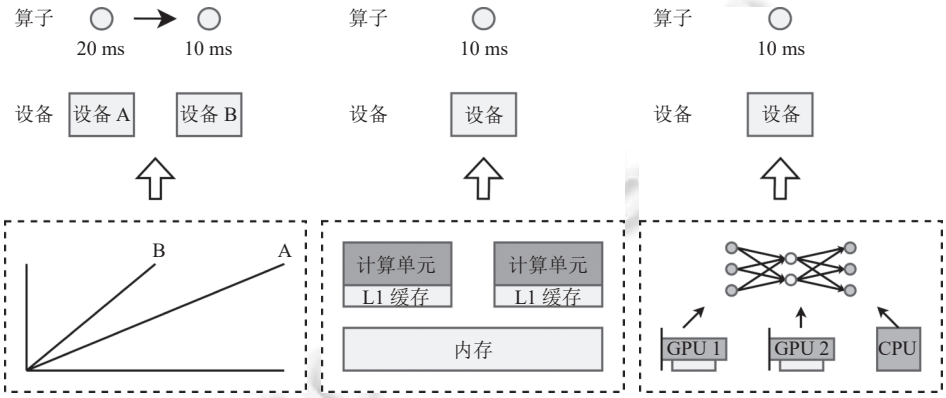
3.2.1 面向稳定深度学习训练配置的监测方法

深度学习框架在深度学习参数与计算资源配置不变时, 通过监测训练阶段若干迭代的运行, 获取在整个深度学习训练阶段中持续有效的算子运行时间信息. AccUDNN[26]选择 5 种批尺寸试运行训练, 用于构建算子性能预测模型. PoocH[25]通过若干迭代的试运行采集算子运行时间以及数据移动时间. vDNN[18]进行若干迭代的试运行, 在每次试运行中尝试不同的内存优化方案组合, 监测其运行性能与内存占用, 并确定最优的优化方案组合. Capuchin[19]、moDNN[21]使用若干种配置运行一次迭代, 采集算子内存占用与计算时间. Tahoe[31]、AutoTM[39]将算子放置于 DRAM 与 NVM, 并监测算子在输入输出数据分别放置于这两类内存时的计算时间. FlashNeuron[29]多次试运行训练, 在每次迭代中首先应用内存换出机制, 并观察其对于训练迭代运行时间与内存瞬时需求的影响. XSP[40]提供面向整个 GPU 软硬件栈的监测机制, 可以分析算子在所有栈层次上的运行时间. ValueExpert[41]捕获每个算子的运行时间, 并分析算子各部分的取值对算子运行时间的影响.

内存使用的监测仅在深度学习训练开始的若干个迭代运行, 因此具有较低的开销. 但随着深度学习训练向云端的迁移, 深度学习训练可能运行于训练集群中, 由深度学习调度框架动态分配与调整训练资源. 深度学习框架可能需要在大量参数、资源配置上进行试运行, 并在运行时采取其他优化方案优化深度学习的内存占用, 在这些情况下, 深度学习训练的运行时间与内存需求均可能发生改变, 频繁监测将产生较大的开销.

3.2.2 面向动态深度学习训练配置的预测方法

深度学习框架面向深度学习训练参数配置 (批尺寸[26]、算子实现[18,23]等配置) 与资源配置 (GPU、内存等配置[42,43]) 动态变化导致的算子运行时间动态变化, 通过构建分析模型, 直接评估新配置下的算子运行时间, 以降低频繁监测产生的开销[26,27,31]. 预测方法如图 5 所示.



(a) 基于比例的预测方法 (b) 基于计算设备模型的预测方法 (c) 基于机器学习的预测方法

图 5 预测方法

(1) 基于比例的预测方法

如图 5(a) 所示, 基于比例的预测方法根据算子运行时间与运行性能与某一配置指标的线性关系, 预测其他配置下的算子运行时间. 例如, 若某一计算设备 B 的每秒浮点数处理操作 (FLOPs) 为当前计算设备 A 的 FLOPs 的 2 倍, 则算子在 B 上应有减半的运行时间. Paleo[27]选取计算设备的每秒浮点数处理操作 (FLOPs) 指标用于预测算子运行时间. AccUDNN[26]监测基准批尺寸下算子的浮点数操作数量、内存占用与 PCIe 传输时间, 并预测其他批尺寸下算子的运行时间. Tahoe[31]基于 CPU 运行的算子在 DRAM 与 NVM 上的运行时间, 预测算子在任意内存比例下的运行时间. Habitat[28]选取 GPU 内存带宽、GPU 时钟频率与 GPU 单次处理的 warp 数量, 预测某一计算设备资源配置下的算子运行时间.

基于比例的预测方法可以用于预测内存交换时间. Werkhoven[44]构建 PCIe 内存交换模型, 通过 LogGP 模型预测算子数据通过 PCIe 总线内存交换所需的时间. Zhang 等人[45]的工作预测算子数据通过 PCIe 总线内存交换所需时间的下界.

基于比例的预测方法通过简单的比例计算预测算子的内存使用状况. 但这一方法过于简化计算设备的计算流程, 实际的内存使用状况受到计算设备的实时性能与利用率的影响. 因此, 比例方法预测算子资源使用状况的效果较为有限.

(2) 基于计算设备模型的预测方法

如图 5(b) 所示, 基于计算设备模型的预测方法模拟专用计算设备, 预测算子的运行时间. DLsim[46]构建 CPU 模拟工具, 分析算子对于 CPU 多级缓存与 DRAM 的使用, DNNMem[47]、Daydream[48]构建 GPU 模拟工具, 模拟执行训练任务的机器指令, 获取算子运行时间、内存占用等信息, 并考虑到内存分配策略, 内存对齐策略等对内存与运算时间的影响. Pei 等人[49]的工作构建对于 GPU 的抽象指令队列模型, 使用算子的 GPU 底层指令构建抽象指令, 通过分析抽象指令的时钟周期, 预测算子的运行时间.

基于计算设备模型的预测方法由于需要详细了解计算设备的运行原理与资源分配特征, 为了更为准确预测算

子的运行时间, 这一方法需要更深层次上刻画计算设备的计算流程, 进而产生更高的分析开销. 此外, 构建计算设备模型需要设计人员对于计算设备具有深入的了解, 具有较高难度.

(3) 基于机器学习的预测方法

如图 5(c) 所示, 基于机器学习的预测方法通过采集算子在特定参数与资源配置下的运行时间, 训练机器学习模型, 预测算子在其他配置的运行时间. Habitat[28]指出卷积等具有可变实现的算子无法通过简单的比例计算进行预测, 并为这些算子构建多层感知机模型, 预测这些算子在不同计算设备上的运行时间. Tahoe[31]构建 3 层神经网络模型, 输入 CPU 最后一级缓存缺失率与迭代周期内指令执行量, 预测算子在数据放置于 DRAM 与 NVM 时的运行时间. Justus[50]、Kaufman[51]输入深度学习算子特征与硬件特征, 构建预测算子在不同 GPU、TPU 等设备上的运行时间的深度学习模型.

由于将计算设备视为黑盒, 基于机器学习的预测方法难以充分结合计算设备的特征, 构建具有更高准确率的机器学习模型. 例如, Justus[50]将 Turing、Pascal 等 GPU 架构名称以字符串形式直接输入模型, 而这一方式难以刻画 GPU 架构变化导致的底层计算特性的变化, 以及这一变化对于算子运行时间的影响.

3.3 内存换入机制

换入机制根据数据依赖关系, 确定已换出内存的数据换入内存的时刻, 保障深度学习训练的正常运行. 深度学习框架通过交叠训练运行与内存交换, 减少等待内存交换的阻塞时间, 保障训练性能. 深度学习框架的换入机制可以分为反馈与前馈两类. 前馈机制还可分为基于数据依赖关系的前馈与基于数据访问时间的前馈. 图 6 展示了各类换入机制, 其中 Op2 与 Op3 的数据需要换入内存, 而内存换入机制将决定 Op2 与 Op3 数据换入内存的时刻, 并在不同程度上影响深度学习训练性能.

3.3.1 反馈机制

如图 6(a) 所示, 反馈机制在深度学习训练需要访问某一内存数据时, 触发数据从外部存储的换入. NVIDIA、AMD 等主流专用计算设备厂商通过统一内存技术对此提供支持, 计算设备与 CPU 内存处于同一内存地址空间, 当某一换出的内存数据被访问时, 专用计算设备将产生缺页中断, 由硬件将内存数据从 CPU 内存换入专用计算设备内存. DRAGON[52]处理 GPU 的缺页中断, 将算子数据从 NVMe SSD 换入到 GPU 内存. Behemoth[53]在深度学习框架层次上实现了内存缓冲区, 在访问数据时将其从 CPU 内存换入到 GPU 内存. 反馈的缺陷在于缺少结合数据依赖关系的数据预取机制. Ganguly 等人[54]的工作分析了 GPU 内存预取器随机预取、顺序预取、与树结构近邻预取的策略, 并指出树结构近邻预取考虑某一内存空间中不同区域内存页面的预取情况, 在这些策略中具有最优的效果. 但这一策略仍然仅能保障某一段内存空间中的最优预取效果, 当某一算子运行结束, 深度学习框架需要进行下一算子的反向传播时, 这些方案无法及时预取此算子内存区域中的页面, 深度学习训练必须等待数据换入完成, 产生极大的阻塞开销.

3.3.2 前馈机制

如图 6(b) 与图 6(c) 所示, 前馈机制在换出的数据被访问之前, 将数据从外部存储预取回内存. 根据触发预取的时刻, 这一机制分为基于数据依赖关系的前馈与基于数据访问时间的前馈.

(1) 基于数据依赖关系的前馈机制

如图 6(b) 所示, 在计算某一算子之前, 触发其数据换入, 实现训练运行与内存交换交换的交叠. vDNN[18]、Layrub[24]交叠算子数据移动与前一个算子的计算. LMS[34]、Dymem[32]设置预取阈值, 提前若干个算子进行内存交换, LMS 的预取阈值可以通过试运行方式确定. 基于数据依赖关系的前馈一定程度上可以及时换入需要访问的数据, 保障深度学习训练训练的运行效率. 仅结合数据依赖关系的前馈机制可能导致数据过早或过晚换入. 如图 6(b) 所示, 若算子计算时间长于数据传输时间, 则数据过早换入到内存中, 降低了内存使用效率; 若计算时间短于传输时间, 则计算完成时, 数据仍未完全换入到内存中, 深度学习训练需要阻塞等待数据换入完成. vDNN[18]假设算子计算时间长于数据交换时间, 要求同步进行训练运行与内存交换, 在算子计算快于内存交换时, 计算需要等待内存交换完成, 将产生较大的同步阻塞开销. vDNN++[55]、HUM[56]等后续工作通过异步进行计算与内存交换, 仅要求所有内存交换在反向传播之前完成, 消除了这一性能瓶颈.

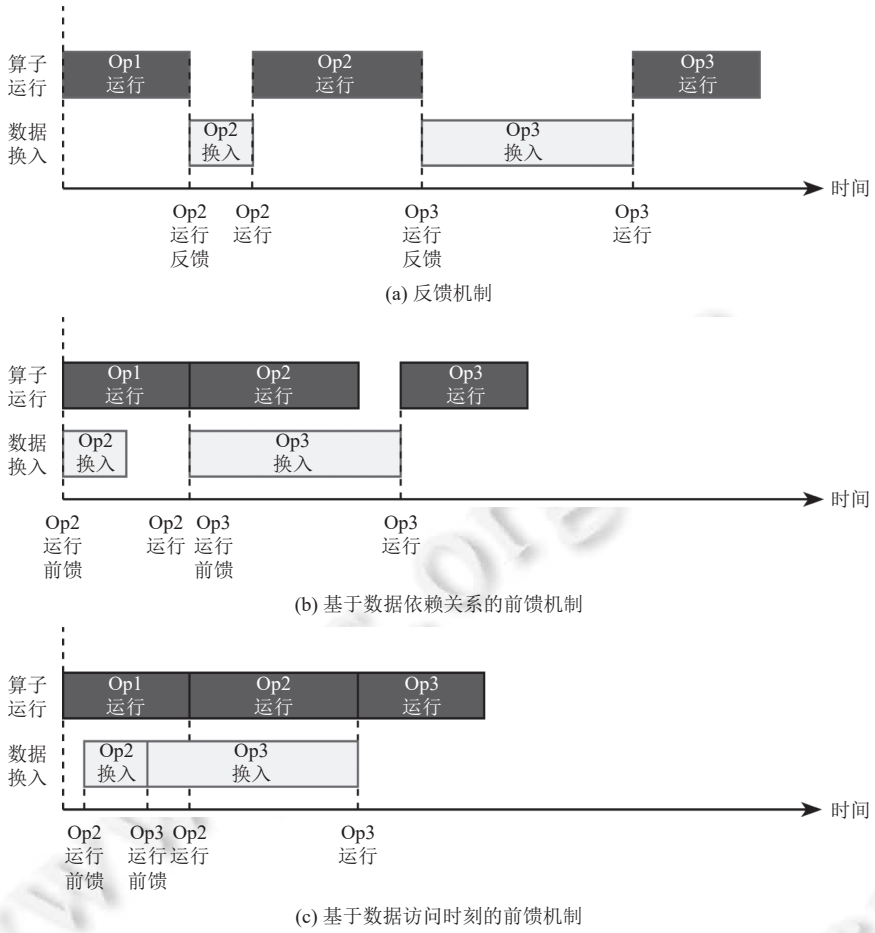


图 6 换入机制

(2) 基于数据访问时间的前馈机制

如图 6(c) 所示, 通过对齐算子运行时间与内存交换时间, 实现在深度学习训练访问数据的同一时刻完成数据换入, 可以消除训练的阻塞时间, 最大化保障深度学习训练性能. 基于数据访问时间的前馈需要考虑算子的计算时间与传输时间. Capuchin[19]根据评估的算子运行时间与数据交换时间, 确定触发预取的最优时刻. Meng 等人[20]的工作基于评估的计算与传输时间, 在计算图中插入数据预取算子. moDNN[21]根据算子计算与数据传输时间, 判断某一数据预取是否可以推迟. Cori[57]从应用程序中提取算子数据访问信息, 在 DRAM 与 NVM 间迁移算子数据, 通过调整迁移频率以满足深度学习训练对算子数据的访问需求. 基于数据访问时间的前馈理论上可以实现算子数据换入完成后立刻访问其数据, 因而可以最大化内存使用效率. 但这一机制要求深度学习框架精确获取各算子的运行时间与算子数据交换时间, 具有较高的实现难度.

3.3.3 内存换入机制的算法描述

算法 2 展示了内存换入机制. 深度学习框架在算子运行前根据算子的运行时间与算子数据交换时间, 确定每个算子需要换入内存的最晚时刻, 通过时间驱动的方式换入算子数据, 以对齐算子换入时间与算子运行时间, 实现内存交换的性能目标.

算法 2. 内存换入机制.

输入: 深度学习训练任务 f, 时刻 i;

输出: 无.

1. while $\exists v' \in f.G.V_{\text{required}}^{(i)}$ ($v' \in V_{\text{storage}} \wedge v' \notin V_{\text{SI}}^{(i)}$) do //所需的算子数据位于外部存储设备, 运用反馈的内存换入机制
2. $V_{\text{SI}}^{(i)} = V_{\text{SI}}^{(i)} \cup \{v'\}$;
3. end
4. $V_{\text{forward}} = \{\}$; //运用前馈的内存换入机制
5. if f.isDataDependencyFeedForward() then //运用基于数据依赖关系的前馈机制
6. V_{forward}=getPostOperators($f.G$, f.forward_thershold);
7. else if f.isDataAccessingTimeFeedForward() then //运用基于数据访问时间的前馈机制
8. V_{forward}= {$v'|v' \in$ getPostOperators($f.G$) $\wedge$ (estimateAccessTime(v')–i)<f.forward_duration};
9. end
10. $V_{\text{SI}}^{(i)} = V_{\text{SI}}^{(i)} \cup V_{\text{forward}}$;
11. swapin($V_{\text{SI}}^{(i)}$);
12. f.execute();

3.4 对比与小结

表 3 展示了部分工作采用的基于数据依赖关系的内存换入机制. 深度学习框架通过利用训练阶段的内存需求时变特征, 基于数据依赖关系, 结合算子运行时间与内存交换时间等因素, 通过反馈与前馈的内存换入机制, 及时换入将被访问的算子数据, 保障深度学习训练的正常运行, 满足内存交换机制的可用性与运行性能目标. 深度学习框架结合深度学习训练运行信息, 面向不同参数与资源配置, 协同使用一系列数据依赖关系与数据访问时间获取方法, 准确高效获取运用内存换入机制所需的信息. 随着深度学习框架开始运用多种运行优化机制, 深度学习训练的内存需求动态变化, 深度学习框架可以进一步探索基于预测的算子运行时间获取方法, 准确高效地实现基于数据访问时间的前馈内存换入机制, 结合多种内存换入机制互为补充, 在内存利用效率、训练信息获取开销与训练性能间进行均衡, 满足深度学习训练的可用性与运行性能目标.

表 3 基于数据依赖关系的内存换入机制

工作	交换粒度	内存类别	数据依赖关系获取	换入机制	数据依赖使用方式	效果
Capuchin[19]	算子数据	GPU内存	监测	基于数据依赖时刻的前馈	根据算子计算时间与内存交换时间, 确定换入时刻	在TensorFlow之上降低85%运行时间
HALO[30]	算子数据	CPU DRAM/HBM	计算图分析	反馈	综合考虑算子计算时间、数据量等, 将数据换入HBM, 提升训练性能	在TensorFlow之上混合利用HBM与DRAM, 提升28.2%性能
Layrub[24]	算子数据	GPU内存	计算图分析	基于数据依赖关系的前馈	交叠算子内存交换与前一算子的计算	在Caffe之上减少内存占用同时, 24.1%运行时间延长
LMS[34]	算子数据	GPU内存	计算图分析	基于数据依赖关系的前馈	交叠算子内存交换与前若干算子的计算	在TensorFlow之上支持大模型训练之外, 7%性能下降
moDNN[21]	算子数据	GPU内存	监测	基于数据依赖时刻的前馈	根据算子计算时间与内存交换时间, 确定换入时刻	基于CUDA实现, 在支持大模型训练之外, 8%性能下降
Tahoe[31]	内存页面	CPU DRAM/NVM	比例预测方法、机器学习预测方法	反馈	将频繁访问的数据换入DRAM, 提升训练性能	在SPECFEM3D等评测基准上实现24%性能提升
vDNN[18]	算子数据	GPU内存	监测	基于数据依赖关系的前馈	交叠算子内存交换与前一算子的计算	与Torch相比, 支持28 GB内存需求的模型训练于12 GB内存上, 18%性能下降

表 3 基于数据依赖关系的内存换入机制 (续)

工作	交换粒度	内存类别	数据依赖关系获取	换入机制	数据依赖使用方式	效果
vDNN++[55]	算子数据	GPU内存	监测	基于数据依赖关系的前馈	交叠算子内存交换与前一算子的交换	在vDNN[18]之上提升60%性能
文献[44]	算子数据	GPU内存	比例预测方法	基于数据依赖时刻的前馈	结合单独获取的数据依赖关系与算子计算时间, 确定内存换入时刻	近90% PCIe传输预测准确率

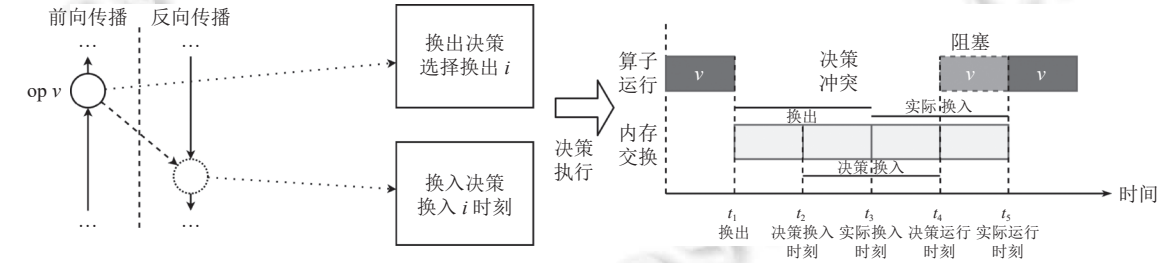
4 效用驱动的内存交换联合决策

深度学习框架需要结合内存换出与换入机制, 利用训练阶段的内存需求时变特征, 进行受深度学习训练效用驱动的内存交换联合决策. 深度学习训练的内存需求时变特征体现在训练运行过程中, 并受到内存交换机制及其优化策略的影响. 单独决策换出与换入无法充分利用内存需求的时变特征, 无法考虑深度学习训练对内存交换的限制, 也无法利用内存交换策略对换出与换入的优化效果, 从而影响内存交换的可用性与性能, 并进而影响深度学习训练的正常运行.

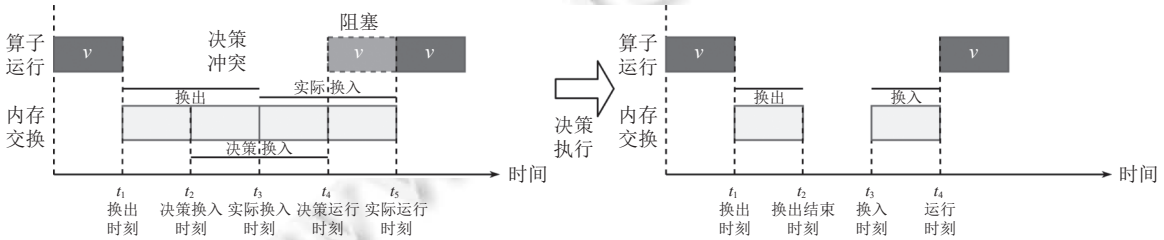
本文第 4.1 节介绍内存交换的两方面影响因素, 第 4.2 节介绍内存交换决策时需要考虑的一系列内存交换优化策略, 第 4.3 节介绍换出与换入的联合决策, 第 4.4 节对本节进行总结.

4.1 内存交换的影响因素

(1) 深度学习训练因素. 单独决策内存换出与换入, 难以有效结合深度学习训练中算子运行顺序的信息. 算子内存先入后出的特征也可体现为后入先出特征, 即最后前向传播的算子最先反向传播, 其数据具有最短的内存占用时间. 在图 7(a) 中, 某一算子经过单独的换出与换入决策, 在 t_1 时刻换出其数据, 在 t_2 时刻换入其数据, 在 t_4 时刻运行此算子, 而在实际运行中, 算子数据在 t_3 时刻完成换出, 随后需要立即换入, 并最终在 t_5 时刻完成换入. 这一决策因为在换出后立即换入, 不仅无法释放内存空间, 而且因为实际换入时刻 t_3 晚于决策换入时刻 t_2, 导致算子的运行被阻塞. 可见, 单独决策换出与换入, 可能将这一算子的数据纳入到交换的范围, 严重影响深度学习训练运行性能.



(a) 深度学习训练因素



(b) 内存交换策略因素

图 7 内存交换决策的影响因素

(2) 内存交换策略因素. 单独决策内存换出与换入, 难以有效利用内存交换的优化策略. 如图 7(b) 所示, 某一算子在没有内存交换优化机制时, 将由于无法及时换入算子数据, 导致深度学习训练的阻塞. 但在应用内存交换优化策略, 降低内存交换开销之后, 其交换时间可以显著缩短, 使换出结束时刻 t_2 不晚于换入开始时刻 t_3, 这一算子的内存数据交换将可以有效降低深度学习训练的瞬时内存需求, 将不会影响深度学习训练的正常运行, 保障深度学习训练运行性能.

4.2 内存交换的优化策略

深度学习训练内存交换的优化策略进行一系列交换数据与交换路径的优化, 以减少内存交换对深度学习训练产生的影响. 由于优化策略可以降低内存交换开销, 原先不可用的交换决策方案可能因使用特定的优化策略而重新可用.

4.2.1 交换数据的优化策略

内存交换需要面对专用计算加速设备有限的数据传输能力. GPU 等设备常用的 PCIe 3.0 x16 总线支持双向 16 GB/s 速率, 这一速率难以支持在百毫秒内交换 GB 级内存数据的需求. 深度学习框架通过压缩交换数据减少内存数据量, 或者增大数据交换粒度以提高交换效率.

(1) 压缩交换数据. 采用无损或有损方式压缩内存数据, 以减少内存交换的数据量. cDMA[58]、Gist[59]利用卷积算子之后的 ReLU 算子输出中包含大量 0 值的特征, 使用压缩稀疏行 (compressed sparse row) 的方式, 以高内存利用效率, 但计算密集的无损压缩方式存储其输出数据, 在不影响深度学习准确度的情况下降低其内存占用. Gist[59] 还利用池化算子之后的 ReLU 算子输出中仅包含 0 或 1 的特征, 使用 1 位代替 32 位的存储形式, 实现 32 倍无损压缩效果. JPEG-ACT[60]对 CNN 算子内存数据使用 JPEG 算法进行有损压缩, 由于 JPEG 算法对整型进行操作, JPEG-ACT 还提出了将 32 位浮点数压缩为 8 位整数的方法. COMET[61]分析有损压缩比率与深度学习准确率之间的关系, 在限制准确率下限的情况下, 尽可能提高压缩比率, 释放内存空间. COMET 应用面向 GPU 优化的 SZ 算法, 与 JPEG 算法相比, 提供了对于浮点数的支持, 更加适应于深度学习训练的需求. Gist[59]、FlashNeuron[29]通过将交换的内存数据存储格式从 32 位浮点数转为 16 位浮点数, 减半内存交换时的数据传输量, 仅产生细微的精度下降. 数据压缩将造成额外的计算开销. 现有工作提出设计专用数据压缩硬件, 避免占用深度学习计算资源, 高效进行数据压缩, 以减少对训练运行的影响[58,60].

(2) 增大交换粒度. 通过增大每次交换的内存数据量, 以提高内存数据通过专用计算加速设备总线的交换效率. 单次交换的数据量过少, 可能难以充分利用总线的传输资源, 影响内存交换效率. 传统的内存交换粒度为 4 KB 的页面, 交换速率仅可达到 80–200 MB/s, 难以满足训练内存数据快速交换的需求[18]. Luley 等人[62]的工作在内存页粒度使用延迟调度方法, 在某一页面需要交换时不立即传输其数据, 而是等待一定时间阈值, 期望累积更多需要传输的页面, 以尝试提高每次交换的数据规模. Zheng 等人[63]的工作构建包含 80 个内存页面的传输集, 在传输集已满时, 或最迟每 20 μs 进行一次传输, 以满足 PCIe 3.0 x16 16 GB/s 的理论传输速率. Yan 等人[64]的工作为操作系统增加了 2 MB 巨页的内存页交换支持. 大部分工作以数百 MB 的算子数据为交换粒度, 可以充分利用传输总线资源, 降低内存交换开销, 满足内存交换的可用性与性能目标.

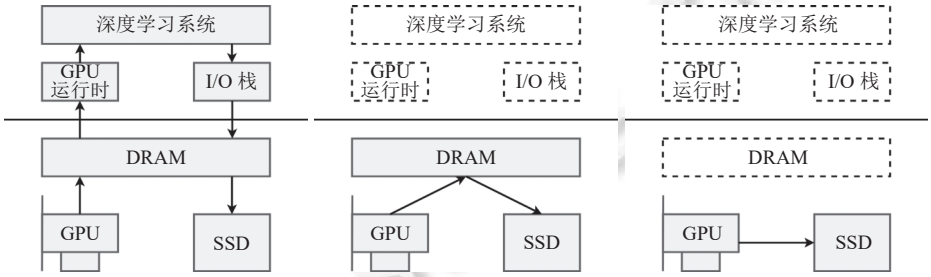
4.2.2 交换路径的优化策略

交换路径的优化策略通过优化内存交换涉及的 GPU、CPU 等专用计算加速设备, PCIe、以太网等传输资源, 以及设备运行时、文件系统、深度学习框架等软件栈, 提高内存交换效率.

图 8(a) 展示了 GPU 与固态硬盘间的内存交换路径. GPU 内存中换出的数据首先需要经 PCIe 总线传输至 CPU 内存, GPU 运行时将数据提交到上层的深度学习框架. 深度学习框架进行文件系统调用, 操作系统 I/O 栈将数据从 CPU 内存写入到固态硬盘中. 这一路径上涉及诸多计算设备与软件栈, 具有较大的优化空间.

(1) 路径长度缩减. 缩短内存交换路径, 以减少内存交换所用的时间. Zheng 等人[63]的工作对 GPU 缺页中断机制进行硬件优化, 减少了 GPU 缺页中断需要通过一系列 CPU-GPU 协作处理带来的开销. NVMMU[65]如图 8(b) 所示, 通过 GPU 映射内存, 将内存区域同时映射到 I/O 栈与 GPU, 消除了交换的内存数据在 CPU 内存中的复制开

销. FlashNeuron[29]如图 8(c) 所示, 使用 P2P 方式直接进行 GPU 内存与 NVMe SSD 之间的内存交换, 将 GPU 内存换出的算子数据直接通过 PCIe 总线传输到 NVMe SSD. Vediappan 等人[66]的工作对内存映射与 P2P 传输进行实验分析, 发现 P2P 传输由于缺少操作系统的页缓存与预读等机制, 在顺序读写上性能显著低于内存映射, SPIN[67]通过为 P2P 传输提供这些机制, 显著提高了 P2P 传输的性能. GPM[68]提供了 GPU 内存与大容量的非易失内存间交换的支持, FlashGPU[69]使用大容量的闪存代替原有的 GPU 内存, 以 GPU 数据存取性能为代价, 消除了 GPU 内存不足的问题. Fastswap[70]、Infiniswap[71]通过 GPU 与 RDMA 网卡之间的 P2P 传输, 提高计算节点的 GPU 内存与其他节点的内存之间的交换效率. Cilo[72]提出新型远端内存节点设计, 通过状态管理在计算节点与内存节点的合理分配, 提高内存交换性能. Kona[73]提出以缓存行粒度进行远端内存交换, 有效避免了远端内存交换中的写放大问题.



(a) 传统路径 (b) 内存映射 (c) P2P

图 8 内存交换路径

(2) 双向利用路径. 通过利用内存交换路径的双向传输能力, 同时进行换出与换入, 可以减少深度学习训练等待内存交换的阻塞开销. ETC[74]通过利用 GPU 的双向 DMA 引擎同时进行内存页面的换出与换入, Yan 等人[64]的工作改造操作系统的内存交换机制, 允许向高性能内存换入热门内存页面的同时, 从中换出非热门页面.

4.3 换出与换入的联合决策

深度学习框架需要进行内存换出与换入的联合决策, 综合考虑深度学习训练因素与内存交换策略因素对内存交换的影响, 构建满足深度学习训练可用性与性能目标的交换方案. 现有成果使用动态或静态的决策方法, 联合进行算子数据换出与换入决策.

(1) 动态决策方法

基于一定规则, 在训练运行阶段即时选择内存数据进行交换. 这一决策方法体现为候选交换数据的过滤. 若交换某部分数据将阻塞深度学习训练, 则此决策方法将部分内存数据排除到交换范围之外. 深度学习框架利用到内存先入后出的算子运行特征, 前向传播中靠前的算子具有更长的内存占用时间, 这一时间更可能长于内存交换时间, 因而交换此算子数据更可能满足深度学习训练的性能目标. Li 等人[36]将访问距离小于特定阈值的算子的内存数据排除到交换范围之外. Capuchin[19]要求算子内存数据交换时间短于算子从其前向传播结束到反向传播开始的间隔时间. Tahoe[31]为深度学习算子进行运行时分析, 对多个算子通过动态规划形成交换决策. Soft2LM[35]监测 CPU 等待内存数据的时间, 并设置内存等待时间阈值, 判断数据是否需要从 NVM 迁移到 DRAM. SuperNeurons[23]根据专用计算设备的内存容量与训练累积内存需求, 为算子计算内存配额, 根据算子对内存的使用决策是否交换其数据. COMET[61]以较低的批尺寸运行训练任务, 在采集若干迭代的运行信息后尝试增大批尺寸, 在运行性能与准确率之间进行均衡. 动态决策不一定形成最优交换决策, 但具有较低的决策开销, 支持训练运行阶段的实时决策.

算法 3 展示了深度学习框架的动态决策方法. 深度学习框架在运行时监测每一时刻需求与交换的算子数据, 若某一被需求算子的数据不处于专用计算加速设备的内存, 或某一算子的换出结束时刻与换入时刻的间隔小于设定的阈值, 则深度学习训练可能需要阻塞等待算子数据. 深度学习框架将此算子排除到交换范围之外.

算法 3. 动态决策方法.

输入: 深度学习训练任务 f, 时刻 i;
输出: 无.

1. $V^{(i)}_{\text{required}} = f.G.V^{(i)}_{\text{required}}$;
2. while $\exists v' \in V_{\text{required}} (v' \in V_{\text{storage}} \vee v' \notin V^{(i)}_{\text{SO}} \vee v' \notin V^{(i)}_{\text{SI}})$ do //此算子的交换导致冲突, 将此算子排除到交换范围之外
3. v'.markNotSwapping();
4. end
5. f.execute();
6. for v' in $V^{(i)}_{\text{SO}}$ do
7. j = estimateSwappingInTimepoint(v');
8. if $j - i < f.\text{interval}_{\text{swap}}$ then //交换此算子将导致冲突, 将此算子排除到交换范围之外
9. v'.markNotSwapping();
10. $V^{(i)}_{\text{SO}} = V^{(i)}_{\text{SO}} \setminus \{v'\}$;
11. end
12. end

(2) 静态决策方法

在深度学习训练前, 形成整个深度学习训练阶段的内存交换计划. 现有深度学习框架使用试运行或整数线性规划的决策方法. 试运行方法通过使用若干参数与资源配置试运行深度学习训练, 记录其运行时间与内存瞬时需求, 最终确定效果最优的内存交换方案. vDNN[18]试运行训练, 决策算子数据交换与算子实现方式, 最终降低深度学习训练内存的瞬时需求. FlashNeuron[29]记录每轮试运行的时间, 并最终选择最小化训练时间的交换方案. PoocH[25]首先交换所有算子的数据, 并通过试运行搜索不会阻塞深度学习训练的算子, 决策交换或重计算其数据. Meng 等人[20]的工作, 与 LMS[34]以及在计算图分析阶段插入换出与换入算子. 整数线性规划决策方法将深度学习训练性能作为目标函数, 将交换约束与交换目标作为约束函数, 规划形成交换决策. AutoTM[39]将决策问题描述为整数线性规划问题, 使用求解器规划内存交换方案. 静态决策可以形成更为优化的交换决策. 其缺陷为存在一定的试运行或求解开销.

算法 4 展示了基于试运行的深度学习框架的静态决策方法. 深度学习框架通过在若干轮迭代中尝试各种内存交换方案, 并确定具有最优性能的内存交换方案. 由于算子在迭代间的稳定性运行特征, 深度学习框架可周期性地运用决策的内存交换方案.

算法 4. 基于试运行的静态决策方法.

输入: 深度学习训练任务 f, 时刻 i;
输出: 无.

1. $\text{count}_{\text{profile}} = f.\text{count}_{\text{profile}}$; //深度学习框架指定的评估迭代数
2. $\text{count}_{\text{step}} = f.\text{count}_{\text{step}}$; //每次迭代中的运行步数
3. P_{SO}={}; //记录每次试运行中的内存交换方案, $|P|=\text{count}_{\text{profile}}$, $p=\{V^{(1)}_{\text{SO}}, V^{(2)}_{\text{SO}}, \ldots, V^{(\text{countstep})}_{\text{SO}}\}$
4. P_{SI}={}; //记录每次试运行中的内存交换方案, $|P|=\text{count}_{\text{profile}}$, $p=\{V^{(1)}_{\text{SI}}, V^{(2)}_{\text{SI}}, \ldots, V^{(\text{countstep})}_{\text{SI}}\}$
5. i=0;
6. while $i < \text{count}_{\text{profile}}$ do //记录每次迭代中的内存换出集合 $V^{(i)}_{\text{SO}}$ 与内存换入集合 $V^{(i)}_{\text{SI}}$

7. $<p_{SO}, p_{SI}> = f$.recordIteration(i);
8. $P_{SO}=P_{SO} \cup \{p_{SO}\}$;
9. $P_{SI}=P_{SI} \cup \{p_{SI}\}$;
10. ++i;
11. end
12. $<p_{fSO}, p_{fSI}>$ = analysis(P_{SO}, P_{SI}); //从试运行结果中
13. f.applicateSwappingPlan(p_{fSO}, p_{fSI}); //选择具有最优性能的内存交换方案

4.4 对比与小结

表 4 展示了现有深度学习框架受深度学习训练效用驱动的内存交换联合决策. 深度学习框架受训练阶段效用驱动, 结合内存换出与换入机制, 针对体现内存需求时变特征的深度学习训练因素, 考虑内存交换机制与内存交换策略等运行时因素的影响, 进行联合内存交换决策. 深度学习框架需要进一步结合训练运行状况, 考虑特定计算资源下特定深度学习训练任务对内存交换的影响, 并利用计算资源、传输资源与存储资源上交换数据与交换路径优化策略, 探索降低内存交换开销的机遇, 根据深度学习训练因素与内存交换策略因素之间的相互影响, 进一步优化内存换出与换入的联合决策, 保障深度学习训练的可用性与性能.

表 4 效用驱动的内存交换联合决策

成果	交换粒度	内存类别	影响因素	联合决策方式	效用目标	效果
AutoTM[39]	算子数据	CPU DRAM/NVM	深度学习训练因素	静态: 整数线性规划	最小化训练时间	将50%–80%数据放置到NVM, 下降27.7%性能
Capuchin[19]	算子数据	GPU内存	深度学习训练因素	动态	最小化阻塞时间	在TensorFlow之上降低85%运行时间, 提高BERT batchsize7倍 (对比TensorFlow)
FlashNeuron[29]	算子数据	GPU内存-SSD	内存交换策略因素	静态: 试运行-过滤	最小化训练时间	在PyTorch之上提升batchsize 12.4–14倍, 因此提升训练吞吐量平均30.3%
Gist[59]	算子数据	GPU内存	内存交换策略因素	静态: 计算图分析	最小化内存瞬时需求	在CNTK之上减少2倍内存占用, 4%性能开销
JPEG-ACT[60]	算子数据	GPU内存	内存交换策略因素	动态	最小化内存交换时间	在DNN评测基准上提升2.4倍内存交换性能, 1.6倍压缩效果
LMS[34]	算子数据	GPU内存	深度学习训练因素	静态:计算图分析	最小化训练时间	在TensorFlow之上以4.9倍更大的批尺寸训练ResNet-50
PoocH[25]	算子输入输出数据	GPU内存	深度学习训练因素	静态: 试运行-过滤	最小化训练时间	在Chainer之上使用16 GB内存训练需求50 GB内存的模型, 38%的性能下降
Tahoe[32]	内存页面	CPU DRAM/NVM	深度学习训练因素	动态	最小化训练时间	在SPECFEM3D等评测基准上实现24%性能提升
vDNN[18]	算子数据	GPU内存	深度学习训练因素	静态: 试运行-过滤	最小化训练时间	与Torch相比, 支持28 GB内存需求的模型训练于12 GB内存, 18%性能下降
文献[20]	算子输入输出数据	GPU内存	深度学习训练因素	静态: 计算图分析	最小化阻塞时间	在TensorFlow之上提高批尺寸2–30倍, 同批尺寸下少于10%的性能下降
文献[36]	算子数据	GPU内存	深度学习训练因素	静态: 试运行-过滤	最小化阻塞时间	与统一内存基础方法相比, 提升性能7.6倍, 最大18.7倍

5 内存交换的应用场景

本节描述内存交换的典型应用场景. 内存交换作为解决内存不足问题的 3 类技术手段之一, 可以与分布式训练、模型压缩其他两类手段共同使用, 实现受限内存资源下的高效深度学习训练. 内存交换可以使用于超大规模深度学习训练场景与边缘深度学习训练场景中.

5.1 大规模分布式深度学习训练场景

使用大规模深度学习集群进行分布式深度学习训练是当前的主流实践[9,10]. 分布式深度学习训练通过数据并行化与模型并行化两种形式, 分别解决大规模数据与大规模模型的挑战. 数据并行化将大规模训练数据切分到各个计算设备上, 分别训练深度学习模型并进行参数同步; 模型并行化将大规模模型切分到各个计算设备上, 保障每个模型切片的内存需求在专用计算设备的内存容量之内.

内存交换机制可以应用于大规模深度学习训练之中, 以降低分布式深度学习训练对于专用计算设备数量的要求. 对于数据并行化, 内存交换机制可以在保障训练性能、模型精度的情况下, 提高批处理尺寸, 加速训练运行. 而对于模型并行化, 内存交换机制可以降低深度学习训练的瞬时内存需求, 从而减少模型并行化所需的专用计算设备数量.

TensorFlow、昇思 (MindSpore)[75]等主流深度学习框架具有对于内存交换机制的支持. TensorFlow 可以利用 IBM 为其提供的大模型支持 (large model support, LMS)[34], 而华为研发的昇思在 1.12 版本中引入了内存交换机制. 使用这些框架运行的深度学习训练可以利用内存交换机制, 降低大规模深度学习训练对于内存的瞬时需求, 保障深度学习训练的运行.

5.2 云边协同深度学习训练场景

云边协同的深度学习训练利用云端与边缘端的算力, 高效支持地理分布的大规模数据集的训练[76]. 边缘端大数据量与强隐私要求的场景特征要求训练数据必须保留在边缘, 产生了边缘端进行深度学习训练的需求, 而边缘端搭载的算力也可以支撑深度学习训练任务. 预训练模型迁移学习与联邦学习是典型的边缘端训练模式. 这两类模式通过协同利用云端与边缘端算力, 在通用模型的基础之上, 训练满足边缘场景需求的深度学习模型.

与传统的深度学习训练集群不同, 边缘端深度学习训练通常仅可提供有限数量的计算设备, 而计算设备的性能与内存容量也相对较低. 在这一场景下, 内存交换机制可以充分利用内存资源, 保障深度学习训练的正常运行.

对于边缘端深度学习训练场景, NVIDIA 的统一内存机制 (unified memory, UM) 是当前可用的工业方案. 统一内存是 NVIDIA 在 2013 年 CUDA 6.0 中引入的内存管理机制. 这一机制将 GPU 内存与 CPU 内存纳入到统一的地址空间, 并提供底层运行时软件与硬件协同的内存交换机制, 在无需修改深度学习框架的情况下, 保障边缘端深度学习训练的运行.

6 展 望

有效结合内存需求的时变特征, 是深度学习训练内存交换机制满足可用性与性能目标的重要原因. 深度学习框架需要深入考虑深度学习训练因素与内存交换策略因素, 以发现通过内存交换降低训练瞬时内存需求, 保障可用性与性能的机遇, 并尝试探索内存交换机制在更为广阔的场景中的应用.

(1) 结合元算子运行特征构建内存换出机制. 现有深度学习框架主要以算子为基本计算单元[16,17]. 近期工作开始引入元算子的概念, 对算子进行进一步抽象. 元算子可以组成常见的深度学习算子. 计图[77]构建了基于元算子的深度学习框架. 谷典典等人[78]在元算子的粒度上进行深度学习框架缺陷检测. 深度学习框架可以对于元算子的运行特征, 如元算子前向与反向传播中的特征, 元算子的内存占用特征等, 进行进一步探索, 构建结合元算子运行特征的内存交换机制.

(2) 面向新型外部存储设备构建内存交换路径优化策略. 现有深度学习框架主要将内存数据换出到 CPU 内存. 随着深度学习训练内存需求不断增高, CPU 内存可能不足以存放换出的内存数据. 部分工作[29,52,53]开始将数据换出到固态硬盘等高容量存储设备中. 固态硬盘由于面向通用应用开发, 缺少面向上层应用特征的针对性优化, 可能构成内存交换瓶颈. 近期关于新型外部存储设备的工作可以应对这一挑战. 这些工作对固态硬盘进行优化, 以提高特定领域应用的性能表现. Willow[79]允许构建运行于固态硬盘之上的程序, 通过将计算任务卸载到固态硬盘运行, 减少数据在固态硬盘与计算设备间移动的开销, 深度学习框架可以考虑将部分高数据量、低计算开销的算子调度到这些设备上运行, 以避免大规模内存交换的开销. OpenChannel SSD[80]、ZNS[81]等技术允许对固态硬盘固件与上

层的文件系统进行定制, 可以减少交换路径上的软硬件开销, 满足内存交换的性能目标. GraphSSD[82]定制了面向图计算的 OpenChannel SSD, 通过将图翻译层和固态硬盘的闪存翻译层合并, 降低在固态硬盘上读写图数据的开销. TOM[83]设计了计算单元与存储单元的融合结构, 允许根据算子的算力与内存需求, 将算子放置于性能更高的 GPU, 或性能较低、内存资源较多的融合计算单元. MC-DLA[84]设计了一种与专用计算设备通过 PCIe 或 NVLINK 连接的内存扩展卡, 允许进行专用计算设备与内存扩展卡间的内存交换. 深度学习框架需要面对新型外部存储设备带来的内存交换优化机遇, 探索内存交换路径的优化策略, 提高内存交换的效率, 满足深度学习训练可用性与性能目标.

(3) 面向推理场景进行内存交换. 现有深度学习框架主要结合深度学习训练阶段的内存需求时变特征, 构建训练阶段的内存交换机制. 未来的工作可以关注内存交换机制在深度学习推理场景的应用. 云边协同中, 进行深度学习推理的边缘设备通常具有较低的计算能力与较少的内存资源, 通常使用经剪裁的深度学习模型, 并存在推理准确率问题[14]. 因此, 深度学习框架可以探索在推理场景中应用内存交换机制, 保障深度学习推理的可用性与性能. 深度学习各应用领域具有最优效果的模型通常具有较大的规模, BERT、GPT-3 等模型的权重数据的内存使用即超过了单个计算设备的内存容量. 因此, 深度学习框架可以对模型权重数据进行交换, 降低推理阶段的瞬时内存需求, 保障推理的可用性与运行性能. 此外, 深度学习推理仅包含前向传播的过程, 某一算子的输出数据仅由其后续算子使用, 而无需为反向传播保留, 因此, 深度学习框架无需为所有算子单独分别分配输入输出数据空间, 而是将这些空间合并, 从而显著降低推理阶段的内存占用.

7 总 结

本文介绍面向深度学习训练内存交换机制, 用以解决深度学习训练显著的内存需求与专用计算加速设备有限的内存供给之间的矛盾. 本文综述深度学习框架结合深度学习训练内存需求时变特征的内存交换机制, 展示深度学习框架如何有效结合深度学习训练内存时变特征, 通过基于算子运行特征的内存换出决策、基于数据依赖关系的换入决策以及效能驱动的联合决策, 满足深度学习训练的可用性与性能. 本文最后结合元算子、新型外部存储设备与推理场景等发展趋势, 对内存交换机制的发展方向进行了展望.

References:

[1] He KM, Zhang XY, Ren SQ, Sun J. Deep residual learning for image recognition. In: Proc. of the 29th IEEE Conf. on Computer Vision and Pattern Recognition. Las Vegas: IEEE, 2016. 770–778. [doi: 10.1109/CVPR.2016.90]

[2] Huang G, Liu Z, van der Maaten L, Weinberger KQ. Densely connected convolutional networks. In: Proc. of the 30th IEEE Conf. on Computer Vision and Pattern Recognition. Honolulu: IEEE, 2017. 2261–2269. [doi: 10.1109/CVPR.2017.243]

[3] Real E, Aggarwal A, Huang YP, Le QV. Regularized evolution for image classifier architecture search. In: Proc. of the 33rd AAAI Conf. on Artificial Intelligence. Honolulu: AAAI Press, 2019. 4780–4789. [doi: 10.1609/aaai.v33i01.33014780]

[4] Amodei D, Ananthanarayanan S, Anubhai R, *et al*. Deep speech 2: End-to-end speech recognition in English and mandarin. In: Proc. of the 33rd Int'l Conf. on Machine Learning. New York: JMLR.org, 2016. 173–182

[5] Dai ZH, Yang ZL, Yang YM, Carbonell J, Le A, Salakhutdinov R. Transformer-XL: Attentive language models beyond a fixed-length context. In: Proc. of the 57th Annual Meeting of the Association for Computational Linguistics. Florence: Association for Computational Linguistics, 2019. 2978–2988. [doi: 10.18653/v1/P19-1285]

[6] Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proc. of the 2019 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Vol. 1. Minneapolis: Association for Computational Linguistics, 2019. 4171–4186. [doi: 10.18653/v1/N19-1423]

[7] Zeiler MD, Fergus R. Visualizing and understanding convolutional networks. In: Proc. of the 13th European Conf. on Computer Vision. Zurich: Springer, 2014. 818–833. [doi: 10.1007/978-3-319-10590-1_53]

[8] Brown TB, Mann B, Ryder N, *et al*. Language models are few-shot learners. In: Proc. of the 34th Int'l Conf. on Neural Information Processing Systems. Vancouver: Curran Associates Inc., 2020. 159.

[9] Kang LY, Wang JF, Liu J, Ye D. Survey on parallel and distributed optimization algorithms for scalable machine learning. Ruan Jian Xue Bao/Journal of Software, 2018, 29(1): 109–130 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5376.htm [doi: 10.

13328/j.cnki.jos.005376]

[10] Zhu HM, Li P, Jiao LC, Yang SY, Hou B. Review of parallel deep neural network. Chinese Journal of Computers, 2018, 41(8): 1861−1881 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2018.01861]

[11] Lepikhin D, Lee H, Xu YZ, Chen DH, Firat O, Huang YP, Krikun M, Shazeer N, Chen ZF. GShard: Scaling giant models with conditional computation and automatic sharding. In: Proc. of the 9th Int'l Conf. on Learning Representations. Austria: OpenReview.net, 2021. 1−23.

[12] Lei J, Gao X, Song J, Wang XL, Song ML. Survey of deep neural network model compression. Ruan Jian Xue Bao/Journal of Software, 2018, 29(2): 251−266 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5428.htm [doi: 10.13328/j.cnki.jos.005428]

[13] Gao H, Tian YL, Xu FY, Zhong S. Survey of deep learning model compression and acceleration. Ruan Jian Xue Bao/Journal of Software, 2021, 32(1): 68−92 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6096.htm [doi: 10.13328/j.cnki.jos.006096]

[14] Chen YJ, Zheng BL, Zhang ZH, Wang Q, Shen C, Zhang Q. Deep learning on mobile and embedded devices: State-of-the-art, challenges, and future directions. ACM Computing Surveys, 2021, 53(4): 84. [doi: 10.1145/3398209]

[15] Ma WL, Peng X, Xiong Q, Shi XH, Jin H. Memory management in deep learning: A survey. Big Data Research, 2020, 6(4): 2020033 (in Chinese with English abstract). [doi: 10.11959/j.issn.2096-0271.2020033]

[16] Abadi M, Barham P, Chen JM, Chen ZF, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M, Kudlur M, Levenberg J, Monga R, Moore S, Murray DG, Steiner B, Tucker P, Vasudevan V, Warden P, Wicke M, Yu Y, Zhang XQ. TensorFlow: A system for large-scale machine learning. In: Proc. of the 12th USENIX Conf. on Operating Systems Design and Implementation. Savannah: USENIX Association, 2016. 265−283.

[17] Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin ZM, Gimelshein N, Antiga L, Desmaison A, Köpf A, Yang EZ, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai JJ, Chintala S. PyTorch: An imperative style, high-performance deep learning library. In: Proc. of the 33rd Int'l Conf. on Neural Information Processing Systems. Vancouver: Curran Associates Inc., 2019. 721.

[18] Rhu M, Gimelshein N, Clemons J, Zulfiqar A, Keckler SW. vDNN: Virtualized deep neural networks for scalable, memory-efficient neural network design. In: Proc. of the 49th Annual IEEE/ACM Int'l Symp. on Microarchitecture. Taipei: IEEE, 2016. 1−13. [doi: 10.1109/MICRO.2016.7783721]

[19] Peng X, Shi XH, Dai HL, Jin H, Ma WL, Xiong Q, Yang F, Qian XH. Capuchin: Tensor-based GPU memory management for deep learning. In: Proc. of the 25th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. Lausanne: ACM, 2020. 891−905. [doi: 10.1145/3373376.3378505]

[20] Meng C, Sun MM, Yang J, Qiu MH, Gu Y. Training deeper models by GPU memory optimization on TensorFlow. In: Proc. of the 2017 Workshop on ML Systems at NIPS. http://learningsys.org/nips17/assets/papers/paper_18.pdf

[21] Chen XM, Chen DZ, Han YH, Hu XS. moDNN: Memory optimal deep neural network training on graphics processing units. IEEE Trans. on Parallel and Distributed Systems, 2019, 30(3): 646−661. [doi: 10.1109/TPDS.2018.2866582]

[22] Zheng BJ, Vijaykumar N, Pekhimenko G. Echo: Compiler-based GPU memory footprint reduction for LSTM RNN training. In: Proc. of the 47th ACM/IEEE Annual Int'l Symp. on Computer Architecture. Valencia: IEEE, 2020. 1089−1102. [doi: 10.1109/ISCA45697.2020.00092]

[23] Wang LN, Ye JM, Zhao YY, Wu W, Li A, Song SL, Xu ZL, Kraska T. SuperNeurons: Dynamic GPU memory management for training deep neural networks. In: Proc. of the 23rd ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming. Vienna: ACM, 2018. 41−53. [doi: 10.1145/3178487.3178491]

[24] Jin H, Liu B, Jiang WB, Ma Y, Shi XH, He BS, Zhao SF. Layer-centric memory reuse and data migration for extreme-scale deep learning on many-core architectures. ACM Trans. on Architecture and Code Optimization, 2018, 15(3): 37. [doi: 10.1145/3243904]

[25] Ito Y, Imai H, Duc TL, Negishi Y, Kawachiya K, Matsumiya R, Endo T. Profiling based out-of-core hybrid method for large neural networks: Poster. In: Proc. of the 24th Symp. on Principles and Practice of Parallel Programming. Washington: ACM, 2019. 399−400. [doi: 10.1145/3293883.3298790]

[26] Guy JR, Liu WT, Wang W, Yao CR, Han JZ, Li RX, Lu YJ, Hu SL. AccUDNN: A GPU memory efficient accelerator for training ultra-deep neural networks. In: Proc. of the 37th IEEE Int'l Conf. on Computer Design. Abu Dhabi: IEEE, 2019. 65−72. [doi: 10.1109/ICCD46524.2019.00017]

[27] Qi H, Sparks ER, Talwalkar A. Paleo: A performance model for deep neural networks. In: Proc. of the 5th Int'l Conf. on Learning Representations. Toulon: OpenReview.net, 2017. 1−10.

[28] Yu GX, Gao YB, Golikov P, Pekhimenko G. Habitat: A runtime-based computational performance predictor for deep neural network training. In: Proc. of the 2021 USENIX Annual Technical Conf. Berkeley: USENIX Association, 2021. 503−521.

[29] Bae J, Lee J, Jin Y, Son S, Kim S, Jang H, Ham TJ, Lee JW. FlashNeuron: SSD-enabled large-batch training of very deep neural networks. In: Proc. of the 19th USENIX Conf. on File and Storage Technologie. Berkeley: USENIX Association, 2021. 387–401.

[30] Han M, Hyun J, Park S, Baek W. Hotness- and lifetime-aware data placement and migration for high-performance deep learning on heterogeneous memory systems. IEEE Trans. on Computers, 2020, 69(3): 377–391. [doi: 10.1109/TC.2019.2949408]

[31] Wu K, Ren J, Li D. Runtime data management on non-volatile memory-based heterogeneous memory for task-parallel programs. In: Proc. of the 2018 Int'l Conf. for High Performance Computing, Networking, Storage and Analysis. Dallas: IEEE, 2018. 401–413. [doi: 10.1109/SC.2018.00034]

[32] Yang DL, Cheng DZ. Efficient GPU memory management for nonlinear DNNs. In: Proc. of the 29th Int'l Symp. on High-performance Parallel and Distributed Computing. Stockholm: ACM, 2020. 185–196. [doi: 10.1145/3369583.3392684]

[33] Zhang JZ, Yeung SH, Shu Y, He BS, Wang W. Efficient memory management for GPU-based deep learning systems. arXiv:1903.06631, 2019.

[34] Le TD, Imai H, Negishi Y, Kawachiya K. Automatic GPU memory management for large neural models in TensorFlow. In: Proc. of the 2019 ACM SIGPLAN Int'l Symp. on Memory Management. Phoenix: ACM, 2019. 1–13. [doi: 10.1145/3315573.3329984]

[35] Giardino M, Doshi K, Ferri B. Soft2LM: Application guided heterogeneous memory management. In: Proc. of the 2016 IEEE Int'l Conf. on Networking. Long Beach: IEEE, 2016. 1–10. [doi: 10.1109/NAS.2016.7549421]

[36] Li LS, Chapman B. Compiler assisted hybrid implicit and explicit GPU memory management under unified address space. In: Proc. of the 2019 Int'l Conf. for High Performance Computing, Networking, Storage and Analysis. Denver: ACM, 2019. 15. [doi: 10.1145/3295500.3356141]

[37] Yu S, Park S, Baek W. Design and implementation of bandwidth-aware memory placement and migration policies for heterogeneous memory systems. In: Proc. of the 2017 Int'l Conf. on Supercomputing. Chicago: ACM, 2017. 18. [doi: 10.1145/3079079.3079092]

[38] Maruf HA, Chowdhury M. Effectively prefetching remote memory with leap. In: Proc. of the 2020 USENIX Conf. on USENIX Annual Technical Conf. Berkeley: USENIX Association, 2019. 58.

[39] Hildebrand M, Khan J, Trika S, Lowe-Power J, Akella V. AutoTM: Automatic tensor movement in heterogeneous memory systems using integer linear programming. In: Proc. of the 25th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. Lausanne: ACM, 2020. 875–890. [doi: 10.1145/3373376.3378465]

[40] Li C, Dakkak A, Xiong JJ, Wei W, Xu LJ, Hwu WM. XSP: Across-stack profiling and analysis of machine learning models on GPUs. In: Proc. of the 2020 IEEE Int'l Parallel and Distributed Processing Symp. New Orleans: IEEE, 2020. 326–337. [doi: 10.1109/IPDPS47924.2020.00042]

[41] Zhou KR, Hao YM, Mellor-Crummey J, Meng XZ, Liu X. ValueExpert: Exploring value patterns in GPU-accelerated applications. In: Proc. of the 27th ACM Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. Lausanne: ACM, 2022. 171–185. [doi: 10.1145/3503222.3507708]

[42] Xiao WC, Bhardwaj R, Ramjee R, Sivathanu M, Kwatra N, Han ZH, Patel P, Peng X, Zhao HY, Zhang QL, Yang F, Zhou LD. Gandiva: Introspective cluster scheduling for deep learning. In: Proc. of the 13th USENIX Conf. on Operating Systems Design and Implementation. Carlsbad: USENIX Association, 2018. 595–610.

[43] Mahajan K, Balasubramanian A, Singhvi A, Venkataraman S, Akella A, Phanishayee A, Chawla S. Themis: Fair and efficient GPU cluster scheduling. In: Proc. of the 17th USENIX Symp. on Networked Systems Design and Implementation. Santa Clara: USENIX Association, 2019. 289–304.

[44] Van Werkhoven B, Maassen J, Seinstra FJ, Bal HE. Performance models for CPU-GPU data transfers. In: Proc. of the 14th IEEE/ACM Int'l Symp. on Cluster, Cloud and Grid Computing. Chicago: IEEE, 2014. 11–20. [doi: 10.1109/CCGrid.2014.16]

[45] Zhang XY, Xiao JM, Tan GM. I/O lower bounds for auto-tuning of convolutions in CNNs. In: Proc. of the 26th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming. New York: ACM, 2021. 247–261. [doi: 10.1145/3437801.3441609]

[46] Chishti Z, Akin B. Memory system characterization of deep learning workloads. In: Proc. of the 2019 Int'l Symp. on Memory Systems. Washington: ACM, 2019. 497–505. [doi: 10.1145/3357526.3357569]

[47] Gao YJ, Liu Y, Zhang HY, Li ZX, Zhu YH, Lin HX, Yang M. Estimating GPU memory consumption of deep learning models. In: Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. New York: ACM, 2020. 1342–1352. [doi: 10.1145/3368089.3417050]

[48] Zhu HY, Phanishayee A, Pekhimenko G. Daydream: Accurately estimating the efficacy of optimizations for DNN training. In: Proc. of the 2020 USENIX Annual Technical Conf. Berkeley: USENIX Association, 2020. 337–352.

[49] Pei ZQ, Li CS, Qin XW, Chen XH, Wei G. Iteration time prediction for CNN in multi-GPU platform: Modeling and analysis. IEEE Access, 2019, 7: 64788–64797. [doi: 10.1109/ACCESS.2019.2916550]

[50] Justus D, Brennan J, Bonner S, McGough AS. Predicting the computational cost of deep learning models. In: Proc. of the 2018 IEEE Int'l Conf. on Big Data. Seattle: IEEE, 2018. 3873–3882. [doi: 10.1109/BigData.2018.8622396]

[51] Kaufman SJ, Phothilimthana PM, Zhou YQ, Mendis C, Roy S, Sabne A, Burrows M. A learned performance model for tensor processing unit. arXiv:2008.01040, 2020.

[52] Markthub P, Belviranli ME, Lee S, Vetter JS, Matsuoka S. DRAGON: Breaking GPU memory capacity limits with direct NVM access. In: Proc. of the 2018 Int'l Conf. for High Performance Computing, Networking, Storage and Analysis. Dallas: IEEE, 2018. 414–426. [doi: 10.1109/SC.2018.00035]

[53] Kim S, Jin Y, Sohn G, Bae J, Ham TJ, Lee JW. Behemoth: A flash-centric training accelerator for extreme-scale DNNs. In: Proc. of the 19th USENIX Conf. on File and Storage Technologies. Berkeley: USENIX Association, 2021. 371–385.

[54] Ganguly D, Zhang ZY, Yang J, Melhem R. Interplay between hardware prefetcher and page eviction policy in CPU-GPU unified virtual memory. In: Proc. of the 46th ACM/IEEE Annual Int'l Symp. on Computer Architecture. Phoenix: ACM, 2019. 224–235. [doi: 10.1145/3307650.3322224]

[55] Shriram SB, Garg A, Kulkarni P. Dynamic memory management for GPU-based training of deep neural networks. In: Proc. of the 2019 IEEE Int'l Parallel and Distributed Processing Symp. Rio de Janeiro: IEEE, 2019. 200–209. [doi: 10.1109/IPDPS.2019.00030]

[56] Jung J, Park D, Do Y, Park J, Lee J. Overlapping host-to-device copy and computation using hidden unified memory. In: Proc. of the 25th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming. San Diego: ACM, 2020. 321–335. [doi: 10.1145/3332466.3374531]

[57] Doudali TD, Zahka D, Gavrilovska A. Cori: Dancing to the right beat of periodic data movements over hybrid memory systems. In: Proc. of the 2021 IEEE Int'l Parallel and Distributed Processing Symp. Portland: IEEE, 2021. 350–359. [doi: 10.1109/IPDPS49936.2021.00043]

[58] Rhu M, O'Connor M, Chatterjee N, Pool J, Kwon Y, Keckler SW. Compressing DMA engine: Leveraging activation sparsity for training deep neural networks. In: Proc. of the 2018 IEEE Int'l Symp. on High Performance Computer Architecture. Vienna: IEEE, 2017. 78–91. [doi: 10.1109/HPCA.2018.00017]

[59] Jain A, Phanishayee A, Mars J, Tang LJ, Pekhimenko G. Gist: Efficient data encoding for deep neural network training. In: Proc. of the 45th ACM/IEEE Annual Int'l Symp. on Computer Architecture. Los Angeles: IEEE, 2018. 776–789. [doi: 10.1109/ISCA.2018.00070]

[60] Evans RD, Liu LF, Aamodt TM. JPEG-ACT: Accelerating deep learning via transform-based lossy compression. In: Proc. of the 47th ACM/IEEE Annual Int'l Symp. on Computer Architecture (ISCA). Valencia: IEEE, 2020. 860–873. [doi: 10.1109/ISCA45697.2020.00075]

[61] Jin S, Zhang CM, Jiang XT, Feng YH, Guan H, Li GP, Song SL, Tao DW. COMET: A novel memory-efficient deep learning training framework by using error-bounded lossy compression. Proc. of the VLDB Endowment, 2021, 15(4): 886–899. [doi: 10.14778/3503585.3503597]

[62] Luley RS, Qiu QR. Optimizing data transfers for improved performance on shared GPUs using reinforcement learning. In: Proc. of the 18th IEEE/ACM Int'l Symp. on Cluster, Cloud and Grid Computing. Washington: IEEE, 2018. 378–381. [doi: 10.1109/CCGRID.2018.00061]

[63] Zheng TH, Nellans D, Zulfiqar A, Stephenson M, Keckler SW. Towards high performance paged memory for GPUs. In: Proc. of the 2016 IEEE Int'l Symp. on High Performance Computer Architecture. Barcelona: IEEE, 2016. 345–357. [doi: 10.1109/HPCA.2016.7446077]

[64] Yan Z, Lustig D, Nellans D, Bhattacharjee A. Nimble page management for tiered memory systems. In: Proc. of the 24th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. Providence: ACM, 2019. 331–345. [doi: 10.1145/3297858.3304024]

[65] Zhang J, Donofrio D, Shalf J, Kandemir MT, Jung M. NVMMU: A non-volatile memory management unit for heterogeneous GPU-SSD architectures. In: Proc. of the 2015 Int'l Conf. on Parallel Architecture and Compilation. San Francisco: IEEE, 2015. 13–24. [doi: 10.1109/PACT.2015.43]

[66] Vediappan A, Mishra D. Analysis of NVMe-SSD to passthrough GPU data transfer in virtualized systems. In: Proc. of the 17th ACM SIGPLAN/SIGOPS Int'l Conf. on Virtual Execution Environments. Virtual: ACM, 2021. 172–185. [doi: 10.1145/3453933.3454023]

[67] Bergman S, Brokhman T, Cohen T, Silberstein M. SPIN: Seamless operating system integration of peer-to-peer DMA between SSDs and GPUs. In: Proc. of the 2017 USENIX Conf. on Annual Technical Conf. Santa Clara: USENIX Association, 2017. 167–179.

[68] Pandey S, Kamath AK, Basu A. GPM: Leveraging persistent memory from a GPU. In: Proc. of the 27th ACM Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. Lausanne: ACM, 2022. 142–156. [doi: 10.1145/3503222.3507758]

[69] Zhang J, Kwon M, Kim H, Kim H, Jung M. FlashGPU: Placing new flash next to GPU cores. In: Proc. of the 56th Annual Design

Automation Conf. Las Vegas: ACM, 2019. 156. [doi: 10.1145/3316781.3317827]

[70] Amaro E, Branner-Augmon C, Luo ZH, Ousterhout A, Aguilera MK, Panda A, Ratnasamy S, Shenker S. Can far memory improve job throughput? In: Proc. of the 15th European Conf. on Computer Systems. Heraklion: ACM, 2020. 14. [doi: 10.1145/3342195.3387522]

[71] Gu JC, Lee Y, Zhang YW, Chowdhury M, Shin KG. Efficient memory disaggregation with infiniswap. In: Proc. of the 14th USENIX Conf. on Networked Systems Design and Implementation. Boston: USENIX Association, 2017. 649–667.

[72] Guo ZY, Shan YZ, Luo XH, Huang YT, Zhang YY. Clio: A hardware-software Co-designed disaggregated memory system. In: Proc. of the 27th ACM Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. Lausanne: ACM, 2022. 417–433. [doi: 10.1145/3503222.3507762]

[73] Calciu I, Imran MT, Puddu I, Kashyap S, Al Maruf H, Mutlu O, Kolli A. Rethinking software runtimes for disaggregated memory. In: Proc. of the 26th ACM Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2021. 79–92. [doi: 10.1145/3445814.3446713]

[74] Li C, Ausavarungnirun R, Rossbach CJ, Zhang YT, Mutlu O, Guo Y, Yang J. A framework for memory oversubscription management in graphics processing units. In: Proc. of the 24th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. Providence: ACM, 2019. 49–63. [doi: 10.1145/3297858.3304044]

[75] Chen L. Introduction to Deep Learning with MindSpore. Beijing: Tsinghua University Press, 2021. 363–394 (in Chinese).

[76] Zhang J, Qu ZH, Chen CX, Wang HZ, Zhan YF, Ye BL, Guo S. Edge learning: The enabling technology for distributed big data analytics in the edge. ACM Computing Surveys, 2022, 54(7): 151. [doi: 10.1145/3464419]

[77] Hu SM, Liang D, Yang GY, Yang GW, Zhou WY. Jittor: A novel deep learning framework with meta-operators and unified graph execution. Science China Information Sciences, 2020, 63(12): 222103. [doi: 10.1007/s11432-020-3097-4]

[78] Gu DD, Shi YN, Liu HZ, Wu G, Jiang HO, Zhao YS, Ma Y. Defect detection for deep learning frameworks based on meta operators. Chinese Journal of Computers, 2022, 45(2): 240–255 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2022.00240]

[79] Seshadri S, Gahagan M, Bhaskaran S, Bunker T, De A, Jin YQ, Liu Y, Swanson S. Willow: A user-programmable SSD. In: Proc. of the 11th USENIX Conf. on Operating Systems Design and Implementation. Broomfield: USENIX Association, 2014. 67–80.

[80] Bjorling M, González J, Bonnet P. LightNVM: The Linux open-channel SSD subsystem. In: Proc. of the 15th USENIX Conf. on File and Storage Technologies. Santa Clara: USENIX Association, 2017. 359–373.

[81] Han K, Gwak H, Shin D, Hwang J. ZNS+: Advanced zoned namespace interface for supporting in-storage zone compaction. In: Proc. of the 15th USENIX Symp. on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2021. 147–162.

[82] Matam KK, Koo G, Zha HP, Tseng HW, Annavaram M. GraphSSD: Graph semantics aware SSD. In: Proc. of the 46th ACM/IEEE Annual Int'l Symp. on Computer Architecture. Phoenix: ACM, 2019. 116–128. [doi: 10.1145/3307650.3322275]

[83] Hsieh K, Ebrahim E, Kim G, Chatterjee N, O'Connor M, Vijaykumar N, Mutlu O, Keckler SW. Transparent offloading and mapping (TOM): Enabling programmer-transparent near-data processing in GPU systems. In: Proc. of the 43rd ACM/IEEE Annual Int'l Symp. on Computer Architecture (ISCA). Seoul: IEEE, 2016. 204–216. [doi: 10.1109/ISCA.2016.27]

[84] Kwon Y, Rhu M. Beyond the memory wall: A case for memory-centric HPC system for deep learning. In: Proc. of the 51st Annual IEEE/ACM Int'l Symp. on Microarchitecture. Fukuoka: IEEE, 2018. 148–161. [doi: 10.1109/MICRO.2018.00021]

附中文参考文献:

[9] 亢良伊, 王建飞, 刘杰, 叶丹. 可扩展机器学习的并行与分布式优化算法综述. 软件学报, 2018, 29(1): 109–130. http://www.jos.org.cn/1000-9825/5376.htm [doi: 10.13328/j.cnki.jos.005376]

[10] 朱虎明, 李佩, 焦李成, 杨淑媛, 侯彪. 深度神经网络并行化研究综述. 计算机学报, 2018, 41(8): 1861–1881. [doi: 10.11897/SP.J.1016.2018.01861]

[12] 雷杰, 高鑫, 宋杰, 王兴路, 宋明黎. 深度网络模型压缩综述. 软件学报, 2018, 29(2): 251−266. http://www.jos.org.cn/1000-9825/5428.htm [doi: 10.13328/j.cnki.jos.005428]

[13] 高晗, 田育龙, 许封元, 仲盛. 深度学习模型压缩与加速综述. 软件学报, 2021, 32(1): 68–92. http://www.jos.org.cn/1000-9825/6096.htm [doi: 10.13328/j.cnki.jos.006096]

[15] 马玮良, 彭轩, 熊倩, 石宣化, 金海. 深度学习中的内存管理问题研究综述. 大数据, 2020, 6(4): 2020033. [doi: 10.11959/j.issn.2096-0271.2020033]

[75] 陈雷. 深度学习与MindSpore实践. 北京: 清华大学出版社, 2021. 363–394.

[78] 谷典典, 石屹宁, 刘譞哲, 吴格, 姜海鸥, 赵耀帅, 马郓. 基于元算子的深度学习框架缺陷检测方法. 计算机学报, 2022, 45(2): 240–255. [doi: 10.11897/SP.J.1016.2022.00240]

高赫然(1997－), 男, 博士生, 主要研究领域为边缘计算, 分布式系统.

李修和(1975－), 男, 博士, 教授, 主要研究领域为复杂电磁环境科学, 体系对抗, 高性能仿真.

吴恒(1983－), 男, 博士, 副研究员, CCF 专业会员, 主要研究领域为容器虚拟化, 边缘计算.

王焘(1982－), 男, 博士, 副研究员, CCF 高级会员, 主要研究领域为软件可靠性, 智能运维, 云计算, 服务计算, 大数据优化.

许源佳(1990－), 男, 博士生, 主要研究领域为资源调度, 分布式系统.

张文博(1976－), 男, 博士, 研究员, 博士生导师, CCF 专业会员, 主要研究领域为云计算, 服务计算.