arXiv:2305.05885v1 [cs.DC] 10 May 2023

P4SGD：面向分布式 FPGA 上广义线性模型的可编程交换机增强型模型并行训练

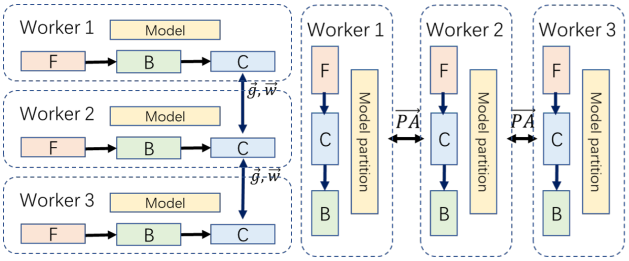
Hongjing Huang、Yingtao Li、Jie Sun、Xueying Zhu、Jie Zhang、Liang Luo、Jialin Li、Zeke Wang*

摘要——广义线性模型（GLM）是现实应用中广泛使用的一类机器学习模型。随着数据规模不断增大，高效地对这些模型进行分布式训练至关重要。然而，现有的分布式训练系统通信成本较高，并且往往使用较大的批量大小来平衡计算与通信，这会对收敛产生不利影响。因此，我们主张构建一种高效的分布式广义线性模型训练系统，力求实现线性可扩展性，同时将批量大小保持在合理的较低水平。为此，我们提出 P4SGD，这是一种分布式异构训练系统，通过分布式 FPGA 之间的模型并行，以及单个 FPGA 内部的前向通信后向流水线并行，高效训练广义线性模型。此外，我们提出一种轻量级、以延迟为中心的交换机内聚合协议，以最大限度地降低由可编程交换机支持的分布式 FPGA 之间 AllReduce 操作的延迟。据我们所知，P4SGD 是首个通过模型并行在分布式加速器之间实现近线性可扩展性的解决方案。我们在八块 Xilinx U280 FPGA 和一台 Tofino P4交换机上实现了 P4SGD。实验表明，与最先进的 GPU 同类方案相比，P4SGD 的收敛速度最高可快 6.5 倍。

索引词—FPGA、P4、广义线性模型、分布式训练系统。

1 引言

机器学习（ML）是一种广受欢迎的方法，无需明确编程即可准确预测结果。以往的大多数工作都专注于加速深度神经网络模型训练，而广义线性模型（GLM），例如线性回归、分类和支持向量机 [1]，，仍然是现实世界中应用最广泛的模型之一 [2]，[3]。由于训练样本数量已从过去的数万增长到如今的数千万 [4]，[5]，而单台机器通常没有足够的计算能力和内存容量，无法针对大量样本高效地进行广义线性模型训练。因此，人们转而采用分布式广义线性模型训练。数据并行和模型并行是分布式训练中两种主流的并行范式。这两种方法都包含三个阶段：前向传播、反向传播和通信，如图 1 所示。



(a) 数据并行 (b) 模型并行

图1：数据并行与模型并行训练的对比，F：前向传播，B：反向传播，C：通信，$\vec{g}$：梯度，$\vec{w}$：模型，PA：部分激活值。数据并行（或模型并行）每次迭代都需要集体通信完整梯度（或一个小批量的激活值）。

数据并行。数据并行将输入数据集横向划分到各个工作节点（即加速器）上。在每次迭代中，每个工作节点使用其本地的整个模型副本（$\vec{w}$），在其子集数据集上进行训练，如图所示如图1a所示。每次迭代结束时，模型（$\vec{g}$）的梯度会通过 AllReduce [6]，[7] 或参数服务器 [8]–[10]在所有工作节点之间进行平均。

模型并行。模型并行会对模型 $\vec{w}$ 和输入数据集进行垂直分区，使每个工作节点在其数据子集上训练模型子集，如图 1b所示。模型并行需要在工作节点之间同步“部分激活值（$\overrightarrow{PA}$）”。

GLM 训练的前向和反向传播过程，每个权重对应的计算量都很低。因此，两种并行方式都需要频繁同步模型（或部分激活值）；当使用更多加速器（如 GPU）时，这会显著影响性能。当模型规模达到数百万个参数时，数据并行需要传输数百万个梯度或模型

• *Hongjing Huang*、*Yingtao Li*、*Jie Sun*、*Xueying Zhu*、*Jie Zhang* 和 *Zeke Wang* 均任职于中国浙江大学人工智能协同创新中心。电子邮箱：*huang hj*、*Li Yingtao*、*jiesun*、*zhux- ueying*、*carlzhang4*、*wangzeke@zju.edu.cn*

• *Liang Luo* 就职于华盛顿大学。电子邮箱：*lian- gluo@cs.washington.edu*

• *Jialin Li* 就职于新加坡的新加坡国立大学。电子邮箱：*lijl@comp.nus.edu.sg*

• *Jie Sun*也隶属于中国阿里巴巴集团。电子邮箱：*sunjie.sun@alibaba- inc.com*

*：通讯作者

通过网络传输，从而产生显著的通信开销。相比之下，模型并行无需交换梯度和模型，只需传输激活。这大幅降低了通信开销，因此在多个加速器上训练大型 GLM（例如全连接层）时，模型并行比数据并行更具潜力。[1] 然而，它仍面临一个严重的问题。

如图1b所示，原始模型并行训练（即基本模型并行训练）需要通过 AllReduce 操作，在前向和反向传播之间汇集部分激活值。这种部分激活值的交换会产生依赖关系，使前向和反向传播无法重叠执行。因此，与数据并行相比，模型并行对吞吐量和延迟都更加敏感 [12]–[15]，，而数据并行对聚合吞吐量更为敏感 [16], [17]。人们普遍认为，模型并行只有在节点内的加速器之间才能实现线性可扩展性 [12]–[15]；这些加速器通过高带宽、低延迟的 NVLink 相连。此外，由于节点间网络带宽有限、网络延迟较高，在模型并行中使用多个加速器进行强扩展（在整个小批量大小不变的情况下，增加用于训练模型的加速器数量所带来的线性加速）仍然颇具挑战。本文提出以下问题：

在分布式加速器之间采用模型并行训练大型 GLM，能否实现强扩展？

为回答这一问题，我们首先提出 P4SGD，这是一种高效的模型并行训练系统，可在训练 GLM 时实现强扩展。[2] P4SGD由部署在可编程交换机上的服务器和部署在 FPGA 上的多个工作节点组成。P4SGD有三项关键创新来解决上述问题：分布式加速器之间的高效模型并行训练（**C1**）、单个加速器内部的前向通信与反向传播流水线并行（**C2**），以及 P4 交换机与 FPGA 之间的超低延迟聚合（**C3**）。

C1：分布式加速器之间的高效模型并行训练。模型并行仅在同一节点内的加速器（如 GPU）之间效果良好，因为这些加速器通过高带宽、低延迟的链路（如 NVLink）连接 [12]–[14]。可扩展性的关键在于实现超低延迟的 AllReduce，并使通信阶段与前向/反向传播阶段重叠。据我们所知，我们率先提出 P4SGD，通过以太网连接的分布式加速器（如 FPGA）之间的模型并行，在 GLM模型（例如全连接层）上实现线性横向扩展训练。

C2：前向传播-通信-反向传播。 FPGA 内的流水线并行。P4SGD 利用一种新的并行维度，即前向传播-通信-反向传播流水线并行，以最大限度提高计算效率。从硬件角度看

1. 这与 Krizhevsky [11], 提出的观察结果一致：由于卷积层每个权重的计算量较大，因此通过数据并行训练卷积层；由于全连接层每个权重的计算量较小，因此通过模型并行训练全连接层。

2. 我们将在未来的工作中将 P4SGD 推广到深度学习模型训练。我们相信，P4SGD 在深度学习模型训练中同样具有巨大潜力。

表1：数据并行（DP）与模型并行（MP）。D：模型维度，M：工作节点数量，S：样本数量，B：小批量大小，BW：工作节点之间的聚合带宽，T_{f_D}：DP 的前向传播时间，T_{f_M}：MP 的前向传播时间，T_l：聚合延迟，T_{b_D}：DP 的反向传播时间，T_{b_M}：MP 的反向传播时间，MB：微批量大小。

	模型 mem.	数据集 mem.	网络 mem.	迭代时间
DP	D	$\frac{S\times D}{M}$	D	$T_{f_D}+\frac{T_{b_D}}{B}+\frac{D}{BW}+T_l$
原始模型并行	$\frac{D}{M}$	$\frac{S\times D}{M}$	B	$T_{f_M}+T_{b_M}+\frac{B}{BW}+T_l$
P4SGD模型并行	$\frac{D}{M}$	$\frac{S\times D}{M}$	B	$\frac{MB}{B}*T_{f_M}+T_{b_M}+\frac{MB}{BW}+T_l$

从视角来看，P4SGD 通过各自独立的硬件资源实现三个不同的执行阶段（即前向传播、通信和反向传播），因此将广义线性模型训练组织为三个执行阶段，以实现阶段间的流水线执行。从软件角度来看，P4SGD 将每个小批量的样本划分为多个更小的微批次，这些微批次依次进入三个执行阶段，彼此之间不受任何依赖关系的影响。因此，P4SGD 可以利用不同微批次的前向/反向传播与通信之间的重叠，同时最大限度地降低模型并行训练的通信开销。

C3：以延迟为中心的交换机内聚合协议。我们设计并实现了一种轻量级、容错的以延迟为中心的交换机内聚合机制，该机制直接协调 P4 交换机与分布式 FPGA，以最大限度地降低每次训练迭代所需 AllReduce 的延迟。此类聚合协议需要分布式 FPGA 与 P4 交换机之间谨慎协调，以从潜在丢包中恢复，同时凭借纯硬件实现，实现超低且稳定的 AllReduce 延迟。

我们在最多八块 Xilinx U280 FPGA开发板上实现 P4SGD 的工作节点 [18]，并在 Tofino P4交换机上实现服务器 [19]。实验结果表明：1）P4SGD 在模型并行模式下，能够在分布式的基于FPGA的加速器之间实现强扩展；2）P4SGD 的收敛速度快于其对应的数据并行系统；3）与分布式图形处理器上的最先进分布式训练系统相比，P4SGD 的收敛速度最高可快 9.3 倍。由于 GPU间通信开销高，这些系统无法实现线性横向扩展训练。

2 背景：SGD硬件并行化

本节简要讨论广义线性模型上数据并行和模型并行训练的硬件实现所具有的有趣特性。通过模型并行或数据并行实现的随机梯度下降（SGD）硬件由三个阶段组成：前向传播、反向传播和通信。这三个阶段各自都需要相应的硬件资源来实现，并且都能进一步挖掘并行性。表1展示了数据并行（“DP”）与模型并行（“原始模型并行”）的比较结果。

2.1 数据并行训练

数据并行沿水平方向将输入数据集划分到各个工作节点上，如图1a所示。每个工作节点维护一份模型的本地副本，并在自己的输入数据集

分区上进行训练。在每次迭代期间，每个工作节点执行三个步骤，如图2a所示。首先，它计算更新后模型与当前小批量中每个样本的点积（前向传播）。其次，它利用该小批量中的点积计算梯度（反向传播）。第三，它将自身的梯度与其他工作节点同步，并通过集体原语 AllReduce 或参数服务器 [8]–[10] 计算更新后的模型。SGD 硬件为前向和反向过程分别配备了定制硬件资源，因此，在对一个样本执行前向过程后，无需等待同一小批量中其他样本的前向结果，即可立即开始反向过程。因此，同一小批量中的样本之间，前向过程和反向过程不存在依赖关系。

估算每轮次耗时。由于这三个阶段各自具有前向传播—通信—反向传播流水线，因此在彼此之间没有依赖关系时，可以并发执行。具体而言，每当前向传播计算出一个样本的损失时，就可以立即开始对该样本进行反向传播，因此这两个阶段可以并发执行，但仍依赖于第三阶段的通信。因此，每次迭代的时间 T_{it} 估算为数据并行（DP）的前向传播时间 T_{f_D} 加上对一个样本进行数据并行（DP）反向传播的时间 $\frac{T_{b_D}}{B}$，再加上通信时间 T_c，如公式1所示。

$$T_{it} = T_{f_D} + \frac{T_{b_D}}{B} + T_c = T_{f_D} + \frac{T_{b_D}}{B} + \frac{D}{BW} + T_l, \quad (1)$$

其中， D 为模型维度， BW 为工作节点之间的聚合带宽， T_l 为聚合延迟。

2.2 模型并行训练

模型并行（MP）将模型沿垂直方向分区并分布到各工作节点上，如图1b所示。每个工作节点维护模型的一个分区，并使用输入数据集的对应分区进行训练。在每次迭代中，每个工作节点都要执行三个步骤，如图2b所示。首先，它计算当前小批量中每个部分样本与部分模型的部分点积（前向传播）。其次，它将小批量中计算出的部分点积与其他工作节点同步，并通过集合通信原语 AllReduce 计算完整点积。因此，需要交换的数据量为 B，即批量大小。第三，每个工作节点使用完整点积计算其梯度部分（反向传播）。

估算每轮次耗时。由于三个阶段在模型方面存在依赖关系，因此它们的执行是串行的。因此， T_{it} 估算为模型并行（MP）的前向传播时间 T_{f_M} 、模型并行（MP）的反向传播时间 T_{b_M} 以及通信时间 T_c之和，如公式2所示。

$$T_{it} = T_{f_M} + T_{b_M} + T_c = T_{f_M} + T_{b_M} + \frac{B}{BW} + T_l \quad (2)$$

2.3 数据并行与模型并行

我们比较了在 M 个工作节点上，数据并行训练与模型并行训练的横向扩展潜力。根据公式 1 和公式 2，我们直观地观察到，数据并行允许前向传播与反向传播重叠执行，但需要在每次迭代期间传输完整的梯度。模型并行每次迭代仅需传输 B 个元素，但由于依赖关系，无法使前向传播与反向传播重叠执行。接下来，我们将讨论两者在横向扩展训练方面的潜力。

数据并行在横向扩展训练中的潜力。 数据并行将一个小批量样本分配给 M 个工作节点，使每个工作节点只需处理 $\frac{B}{M}$ 个样本，这意味着前向传播时间可以缩短 M 倍。然而，其通信时间保持不变，因为需要交换的数据量没有变化。不过在实践中，通信时间会增加。更大的 M 很容易使数据并行受到通信限制，尤其是在每个权重的计算量相对较低的 GLM 上。另一个缺点是，数据并行无法支持超出工作节点内存容量的大型模型。

模型并行在横向扩展训练中的潜力。模型并行将模型和数据集分配给 M个工作节点，使每个工作节点只持有模型的一个分区，并将前向和反向传播时间缩短 M 倍。此外，模型并行只需与其他工作节点交换 B 个元素，产生的开销可以忽略不计。[3] 因此，模型并行可以轻松扩展，以支持 M 个工作节点参与训练。

3 P4SGD 系统概述

3.1 设计目标与总体架构

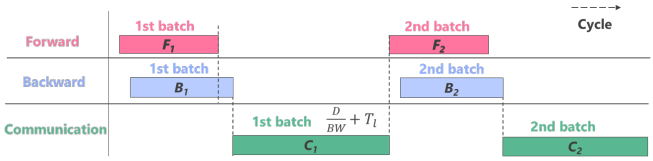
设计 P4SGD 时，我们牢记三个目标。

G1：实现高效的横向扩展训练。随着 GLM 的模型大小不断增加，在内存和计算能力有限的单个工作节点上训练模型并不合适。因此，自然的做法是让多个工作节点并行训练同一个模型。然而，实现线性可扩展性颇具挑战，因为 GLM 每个权重对应的计算量较少，导致通信开销难以在 CPU/GPU 上摊销。

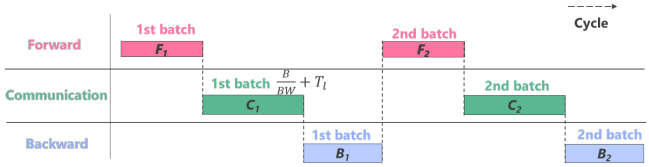
G2：最大限度地重叠前向和反向传播计算。基于原生模型并行的随机梯度下降硬件，由于前向和反向传播之间存在固有依赖，无法实现二者的重叠，导致计算资源利用率低。因此，最大限度地重叠前向和反向传播，可以显著缩短计算时间，进而提升模型并行训练的效率。

G3：最小化通信延迟。即使模型并行训练每次迭代所需的网络流量仅为 B 个元素，网络延迟 T_l 仍可能阻碍线性可扩展性，尤其是在 M 较大时。这是因为， M 个工作节点会使每次迭代的计算时间缩短至原来的 M 分之一，

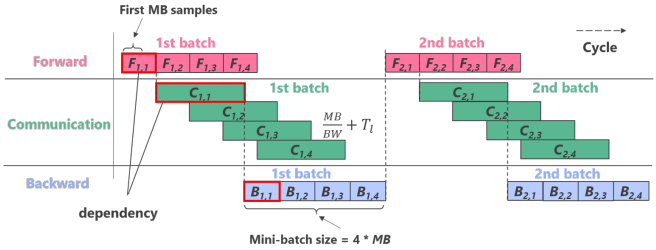
3. 尽管与 B 元素相关的延迟很小，但基本网络延迟 T_l 仍然很大。



(a) 数据并行训练



(b) 原始小批量模型并行训练



(c) 工作节点内的微批量前向-通信-反向流水线并行训练 ‘

图 2：硬件实现的 GLM 训练中数据并行与模型并行的比较，这促使我们采用前向传播—通信—反向传播流水线并行。$F_{i,j}$：第 i 个小批量中第 j 个微批次的前向传播，$B_{i,j}$：第 i 个小批量中第 j 个微批次的反向传播，C：通信，MB：微批量大小，BW：网络带宽，T_l：网络固定延迟

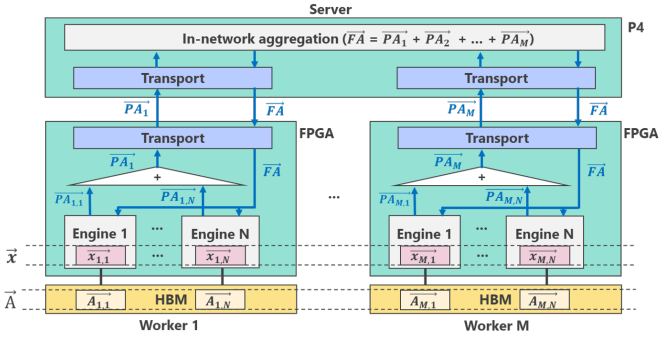


图 3：P4SGD 由 M 个工作节点和一台服务器组成，每个工作节点包含 N 个计算引擎。P4SGD 在分布式工作节点之间实现模型并行，并在单个工作节点的计算引擎内部实现前向传播-通信-反向传播流水线并行。因此，P4SGD 首次实现了分布式加速器之间的线性可扩展性。

这表明有更大比例的时间消耗在通信上。因此，最大限度地降低工作节点之间通信原语 AllReduce 的延迟，将显著有利于横向扩展训练。

总体架构。 为实现上述三个目标，我们提出了 P4SGD，这是一种由 P4 交换机增强的硬件训练系统，可通过在分布式 FPGA 上进行模型并行，高效扩展 GLM 训练。具体而言，如图 3 所示，P4SGD 由 M 基于FPGA 的工作节点和一台由 P4 交换机增强的服务器组成。所有这些

算法 1： 模型并行训练

定义： E：训练轮次数量，
M：工作节点数量，
$A_i[j]$：第 i 个工作节点中的第 j 个样本分区，
PA_m：第 m 个工作节点中的部分激活向量，
b_i：第 i 个样本的标签值，
$\vec{x_i}$：第 i 个工作节点中的局部模型向量，
γ：学习率。

基于 P4 交换机的服务器：
向所有工作节点提出问题 $start$；
从 $e = 1$ **到** E **执行**
  用于$(i = 0;\ i < S;\ i+ = MB)$ **执行**
    Pull $\overrightarrow{PA_m}$ 从所有工作节点；
    $FA = \overrightarrow{PA_1} + ... + \overrightarrow{PA_M}$；
    Push $\overrightarrow{FA}$ 发送给所有工作节点；
  end
结束
基于FPGA的工作节点 m (1, ..., M)：
加载训练数据集 $\vec{a}$ 的第 m 个分区 $\overrightarrow{a_m}$；
初始化模型第 m 个分区 $\vec{x_m}$ 的 $\vec{x}$；
对于 $e = 1$ **到** E **执行**
  对于$(i = 0;\ i < S;\ i+ = B)$ **执行**
    $g_m = 0$;/* 将部分梯度置零 */
    循环$(j = 0;\ j < B;\ j+ = MB)$ **执行**
      /* 阶段 1：前向传播 */
      #pragma 在硬件中并行
      循环 $(k = 0;\ k < MB;\ k++)$ **执行**
        int32 $t = i + j + k$;
        int32 $\overrightarrow{PA_m}[k] = \overrightarrow{A_m}[t] \cdot \vec{x_m}$;
      end
      /* 阶段 2：通信 */
      推送部分激活向量 $\overrightarrow{PA_m}$ 到该地点服务器;
      提取完整激活向量 $\overrightarrow{FA}$ 来自服务器;
      /* 阶段 3：反向传播 */
      #pragma 在硬件中并行执行
      对于 $(k = 0;\ k < MB;\ k++)$ **执行**
        int32 $t = i + j + k$;
        int32 $scale[k] = \gamma \times df(\overrightarrow{FA[k]}, b_{i+k})$;$\overrightarrow{g_m} += scale[k] \times \overrightarrow{A_m}[t]$;
      end
    结束
    $x_m = \vec{x_m} - \vec{g_m}/B$;
  end
结束

M 工作节点以锁步方式完成每次迭代，从而在相同的模型 $\vec{x}$ 上高效训练。每个工作节点通过模型并行，在数据集的一个子集上训练模型的一个子集。在每次迭代中，训练模型子集 $\vec{x_m}$ 的第 m 个工作节点会向基于 P4 交换机的服务器发送一个网络数据包，其中包含其部分聚合结果 $\overrightarrow{PA_m}$；服务器将这些结果聚合后，再向所有工作节点广播一个包含 $\overrightarrow{FA}$ 的网络数据包，以便进一步进行反向传播计算。

P4SGD 的一个关键思想是将每个小批量样本划分为多个更小的微批次（**G2**），使随机梯度下降硬件能够利用微批次间（即小批量内）的并行性，最大限度地重叠前向/反向传播与通信，同时仍保留同步 SGD 所需的先后顺序（第 3.2 小节），如算法 1 所示。另一个关键思想是轻量级、容错、以延迟为中心的交换机内聚合（**G3**），它使 P4 交换机与分布式 FPGA 直接交互，以尽可能降低 AllReduce 的延迟（第 3.3 小节）。与此同时，通信开销也随之降低，这能够显著提升 P4SGD 的可扩展性（**G1**）。

3.2 基于 FPGA 的工作节点架构

算法 1 展示了 P4SGD 下第 m 个基于 FPGA 的工作节点的详细执行流程。首先，它加载数据集 $\overrightarrow{a_m}$ 的分区（第 11 行），并初始化模型 $\overrightarrow{x_m}$ 的分区（第 12 行）。该算法迭代执行 E 个训练轮次（第 13 行）。在每个训练轮次中，它扫描输入数据集的分区，在每次迭代中处理一个包含 B 个样本的小批量（第 14 行）。在一个小批量内，我们将部分梯度 $\overrightarrow{g_m}$ 置零（第 15 行）。随后，它持续累积来自一个包含 MB 个样本的微批次的梯度 $\overrightarrow{g_m}$，具体过程如下文所述的三个阶段。

阶段 1：前向传播。第 m 个工作节点读取 MB 个部分样本及其部分模型 $\overrightarrow{x_m}$，并计算包含 MB 个元素的部分激活值 （$\overrightarrow{PA_m}$）（第 17—21 行）。

阶段 2：通信。m 个工作节点发送 $\overrightarrow{PA_m}$至服务器进行聚合，有效载荷包含 MB 个元素。随后，它等待相应的完整激活 （$\overrightarrow{FA}$），以进行后续的反向传播（第 22—23 行）。

阶段 3：反向传播。第 m 个工作节点根据样本微批次计算部分梯度，并将其累加到部分模型 $\overrightarrow{x_m}$ （第 25—29 行）。然后，使用平均梯度 $\overrightarrow{g_m}$ 更新部分模型 $\overrightarrow{x_m}$。

3.2.1 并行性分析

图 2c 展示了工作节点内微批次流水线并行训练的三个阶段之间的依赖关系。第一个小批量的第一个微批次 $F_{1,1}$ 完成前向传播计算后，无需等待其他微批次完成，即可在该微批次内启动网络集体操作，直接进入通信阶段。当第一个批次 $B_{1,1}$ 从服务器接收到相应的完整激活向量后，即可直接进入第三阶段。我们观察到，同一小批量内的微批次之间（例如 $F_{1,1}$ 和 $F_{1,2}$）不存在依赖关系，因此可以在小批量内重叠通信与前向/反向传播计算。

讨论。一个潜在的局限是，每次仅使用少量样本（例如 8 个）组成的微批次进行训练，会降低计算并行度，进而导致现代 CPU/图形处理器上的计算能力和网络带宽利用不足，因为它们依赖软件实现。然而，依赖纯硬件实现的 P4SGD（例如 FPGA 和 P4）更能适应小微批次训练，而不会牺牲计算吞吐量。

3.2.2 估算 *P4SGD* 的每次迭代时间

P4SGD 支持不同微批次之间前向和反向传播计算的重叠，如图 2c 所示，因此我们估算每次迭代的耗时 T_{it} 为一个微批次的前向传播时间 $\frac{BW}{B} \times T_{f_M}$ 加上通信时间 T_c，再加上一个小批量的反向传播时间 T_{b_M}，如公式 3 所示。与图 2b 所示的小批量模型并行训练相比，所提出的微批次训练能够最大限度地实现前向和反向传播的重叠。

$$T_{it} = \frac{MB}{B} \times T_{f_M} + T_{b_M} + \frac{MB}{BW} + T_l \tag{3}$$

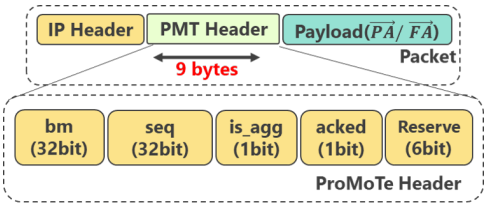


图 4：P4SGD 数据包格式

3.3 P4 交换机-FPGA 集合操作

根据上述分析，通信延迟 T_c对整体训练时间至关重要。此外，P4SGD 需要在每次迭代中对 MB 个元素执行一次 *AllReduce* 操作，其中 MB 在我们的训练过程中通常较小。因此，P4SGD 要求对小型有效载荷执行 *AllReduce*操作时具有极低的延迟。

在网络交换机中聚合部分激活值有两大延迟优势。首先，由于交换机位于网络内部，我们的方法避免了往返聚合服务器所需的额外网络跳数，从而基本上将激活聚合的延迟降低至“亚往返时延”。其次，交换机数据平面专为快速、可预测的数据包处理而设计。包括新一代可编程交换机 [20]–[23]，在内的商用网络交换机，能够持续在几百纳秒内处理数据包——相比之下，即便是配备远程直接内存访问或采用内核旁路网络的服务器，处理速度也明显更慢。

然而，传统的交换机内聚合方法 [16], [17], 要求终端主机服务器在网内聚合之前准备数据包，因此会因不稳定的软件处理开销和较长的 PCIe 延迟而产生较高的通信延迟。此外，这些方法引入了影子副本机制，以便在对大型消息执行 AllReduce 时提高吞吐量；但由于确认发送较晚，这些方法不利于降低延迟。为此，我们设计并实现了一种轻量级、容错且**以延迟为中心的** 交换机内聚合方案，通过 P4交换机与分布式 FPGA 直接协同工作，最大限度地降低每次训练迭代所需 AllReduce 操作的延迟。

在 P4SGD 中，我们通过以 FPGA 为中心的重传机制处理数据包丢失。与 SwitchML [16][4], 采用的方法相比，我们的方案无需在交换机上为聚合结果保留影子副本，并显著降低了交换机资源使用量。在相同的资源预算下，SwitchML 能支持的未完成聚合操作数只有我们方案的一半。

为解决因数据包丢失而导致聚合结果丢失的问题（数据包中包含 $\overrightarrow{FA}$），我们的核心思路是再进行一轮通信，由工作节点发起并重传，以确认完整激活向量已接收。只有当交换机收到所有工作节点针对某项操作发来的确认后，才能安全地清除聚合结果。

具体而言，如算法2所示，交换机仅维护一份聚合后的激活值副本，agg。为了

4. SwitchML 是一种广泛用于分布式训练的交换机内聚合方法。它采用影子副本机制来优化吞吐量。

为分别跟踪已接收的激活和确认，交换机存储两组计数器，$agg\ count$ 和 $ack\ count$。由于工作节点会在数据包丢失时重传数据包，因此维护两组位图，$agg\ bm$ 和 $ack\ bm$，以检测重复项。我们还为 P4SGD 数据包头添加了额外字段，如图4所示：bm 是一个位图，其中源工作节点的索引被置为 1；seq 是 P4交换机中的聚合时隙索引；$is\ agg$ 表示数据包用于聚合还是确认；$acked$ 是一个占位符，供交换机指示其已收到该操作的所有确认。

算法2： 包含
不可靠传输处理的交换机聚合逻辑

初始化： N：聚合槽位数量
$\overrightarrow{W}$：工作节点数量
$agg[N] := \{\overrightarrow{0}\}$
$agg_count[N], agg_bm[N] := \{0\}$
$ack_count[N], ack_bm[N] := \{0\}$

1 接收 数据包*(is_agg, seq, bm, $\overrightarrow{PA}$, FA)*:
2 **如果** is_agg **则**
3 **如果** $agg_bm[seq] \& bm = 0$ **则**
4 $agg_count[seq] \leftarrow agg_count[seq] + 1$;
5 $agg_bm[seq] \leftarrow agg_bm[seq] | bm$;
6 $\overrightarrow{agg}[seq] \leftarrow \overrightarrow{agg}[seq] + pkt.\overrightarrow{PA}$;
7 **如果** $agg_count[seq] = W$ **则**
8 $ack_count[seq] \leftarrow 0$;
9 $ack_bm[seq] \leftarrow 0$;
10 **end**
11 **end**
12 **如果** $agg_count[seq] = W$ **那么**
13 $pkt.\overrightarrow{FA} \leftarrow \overrightarrow{agg}[seq]$;
14 将 pkt转发给所有工作节点;
15 **end**
16 **end**
17 **else**
18 **if** $ack_bm[seq] \& bm = 0$ **则**
19 $ack_count[seq] \leftarrow ack_count[seq] + 1$;
20 $ack_bm[seq] \leftarrow ack_bm[seq] | bm$;
21 **如果** $ack_count[seq] = W$ **则**
22 $agg_count[seq] \leftarrow 0$;
23 $agg_bm[seq] \leftarrow 0$;
24
25 $\overrightarrow{agg}[seq] \leftarrow \overrightarrow{0}$;
26 **end**
27 **如果** $ack_count[seq] = W$ **则**
28 将 pkt转发给所有工作节点;
29 **end**
30 **end**
31 **结束**

我们相应地扩展了工作节点协议（算法3）。向交换机发送部分激活时，工作节点会将其节点位图添加到数据包中，并标明该数据包用于聚合（第5行）。收到完整激活后，工作节点会向交换机发送确认（第22–23行）。不过，只有收到交换机发来的确认确认后，工作节点才会重新启用用于聚合的槽位（第26–29行）。为处理数据包丢失，工作节点在发送每个数据包后启动定时器（第11、24行），并在定时器到期后重新传输该数据包（第31–34行）。收到对应的完整激活（第20行）或确认确认（第28行）后，定时器便会取消。

算法 3： 工作节点端聚合逻辑
采用不可靠传输处理

初始化： N：聚合时隙数
<escape>$unused[N]$</escape>:= {true}
$seq := 0$, $bm :=$ 工作节点 索引
<escape>PA</escape>:

1 发送*pa* 数据包*(*
2 **如果** $unused[seq]$ **则**
3 $unused[seq] \leftarrow$假;
4 $pkt.seq \leftarrow seq$; $pkt.\overrightarrow{PA} \leftarrow \overrightarrow{PA}$;
5 $pkt.bm \leftarrow bm$; $pkt.is_agg \leftarrow$ 真;
6 $seq \leftarrow seq + 1$;
7 **如果** $seq = N$ **则**
8 $seq \leftarrow 0$;
9 **end**
10 将 pkt转发到交换机;
11 启动定时器(pkt);
12 返回 true;
13 **end**
14 **else**
15 返回 false;
16 **end**
17 **结束**
18 接收 数据包*(is_agg, seq, bm, $\overrightarrow{PA}$, $\overrightarrow{FA}$)*:
19 **如果** $pkt.is_agg$ **则**
20 取消定时器(pkt); 将 $pkt.\overrightarrow{FA}$ 前向传播至反向传播;
21
22 $pkt.is_agg \leftarrow$ false; $pkt.bm \leftarrow bm$;
23 向前转发 pkt至交换机;
24 启动定时器(pkt);
25 **end**
26 **else**
27 $unused[pkt.seq] \leftarrow$ true;
28 取消定时器(pkt);
29 **end**
30 **结束**
31 超时时*(*数据包*)*:
32 转发 pkt到交换机;
33 启动定时器(pkt);
34 **结束**

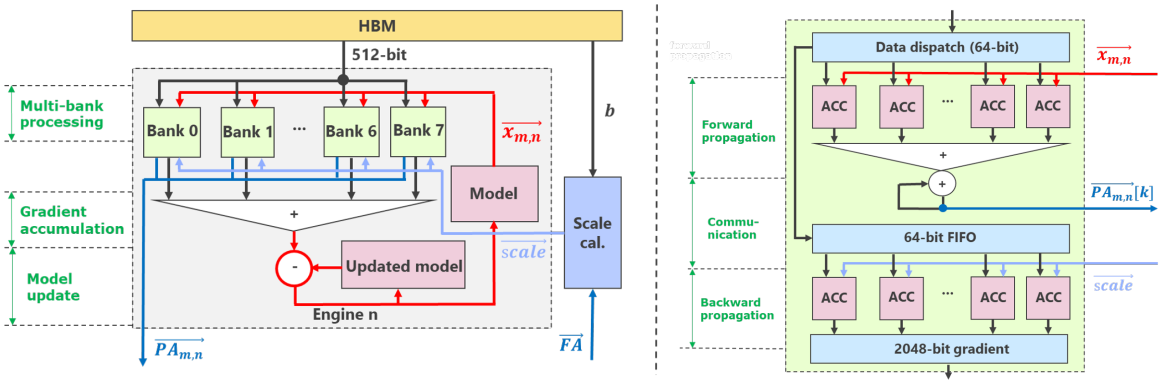
4 P4SGD 的实现

我们使用一台基于 P4 交换机的服务器和M 个基于 FPGA 的工作节点实现了 P4SGD。下文将介绍基于 FPGA 的工作节点和基于 P4 交换机的服务器的实现细节，如图5所示。

4.1 工作节点的硬件设计

基于FPGA的工作节点的目标有三方面。首先，它最大化处理并行度（**E1**），这不仅需要强大的处理能力，还需要广义线性模型训练所需的高内存带宽。其次，它支持微批次流水线并行训练，使前向和反向传播计算能够重叠，从而提高计算流水线利用率（**E2**）。第三，它需要将通信与前向/反向传播计算重叠，以摊销FPGA间通信的负面影响（**E3**）。

我们使用 Verilog 语言，在配备高带宽存储器的 Xilinx Alveo U280 [18] FPGA 板上实现每个工作节点。为了实现 **E1**，我们采用多引擎设计，将分配给该工作节点的模型子集均匀划分给 N 个引擎，其中 N 在编译时参数化。这样，所有引擎以锁步方式训练，每个引擎只需训练该子集的一小部分。例如，$\overrightarrow{x_1}$，2 对应于与第一个工作节点的



（a）第 m 个工作节点中第 n个引擎的详细设计

（b）第 n-个工作节点中第 m 个引擎的详细设计，可实现前向-通信-反向流水线并行

图 5：P4SGD 的每个工作节点包含 N 个引擎。每个引擎有 8 个存储体，用于处理由 8 个样本组成的微批次。

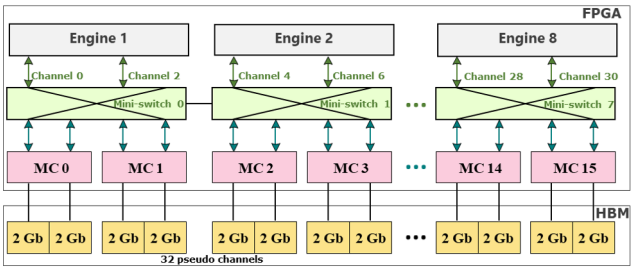


图 6：16 个 HBM 通道与 8 个引擎之间的关系，每个引擎占用 4 个 HBM 伪通道。

第二个引擎相关联的模型部分。[5] 每个模型部分最多容纳 256K 个权重，[6] 并由片上存储器实现，因此每个工作节点支持 2M 个权重。因此，如图 3 所示，每个工作节点由一个高带宽存储器子系统（第 4.1.1 节）、 N 个引擎（第 4.1.2、4.1.3 节）组成。

4.1.1 内存子系统的硬件设计

内存子系统采用包含 32 个内存通道的 HBM 子系统，以支持 32 路独立的 256 位内存访问。由于 FPGA 资源有限，一块 FPGA 板最多只能容纳 N=8 个计算引擎；每个计算引擎可占用连续的四个 HBM 通道，为其负责的数据集子集提供充足的内存带宽和容量（8Gb）。图 6 展示了 32 个 HBM 通道与 8 个计算引擎之间的关系。由于每个计算引擎的设计目标是每周期处理 512 位数据流，因此每个计算引擎会组合两个 256 位 AXI 接口来访问 HBM 中的数据。

4.1.2 计算引擎的硬件设计

基于先进的 GLM 加速器 MLWeaving [24]（支持高效的低精度训练）构建，每个计算引擎均经过扩展，以支持在分布式 FPGA 上同步执行 GLM 训练。为实现第 4.1 节中的 **E2**，我们采用了一种采用多银行设计，使每个计算引擎能够并发训练一个由 MB=8 个样本组成的微批次，每个银行存放一个样本。因此， MB 个样本会并发流入计算流水线。

每个计算引擎的硬件设计包含三个阶段：“多银行划分”、“梯度累积”和“模型更新”。在“多银行划分”阶段，512 位输入数据流流入 8 个银行，每个银行从同一样本中读取 64 位数据，即一个样本的一个特征中的一位。每个阶段的详细设计见第4.1.3小节。每个银行的输出是一个 2048 位梯度（即来自 64 个特征的 64 个 32 位梯度元素）。在“梯度累积”阶段，我们实例化了 64 个逐元素的 8 输入加法树，以汇总来自所有 8 个银行的梯度。与 MLWeaving 不同，我们使用数字信号处理器而非查找表来构建加法树。加法树的每一层都能将三个数相加，从而节省资源并缩短计算时间。在“模型更新”阶段，将上述 MB=8 个样本的聚合梯度累加到“更新后的模型”中。将每个小批量中最后一个微批次的梯度累加到“更新后的模型”时，我们还会使用写入“更新后的模型”的相同值更新架构模型 ($\vec{x}$)。

4.1.3 银行的硬件设计

图5b展示了每个存储组的详细信息。根据分布式随机梯度下降算法，每个存储组由三个阶段组成。为实现第4.1节中的 **E3**，我们在“通信”阶段每次发送 MB 个样本的“部分激活值”，以支持GLM上的微批次模型并行训练。

每个存储区的硬件设计由三个阶段组成：“前向传播”、“通信”和“反向传播”。在“前向传播”阶段，我们实例化 64 个位串行乘法器 [25], [26]，以消耗 64 位数据流，每个特征对应一位。同时，64 位数据流也会被输入 64 位“先进先出队列”。每个位串行乘法器输出一个 32 位计算结果，并将其直接输入全流水线加法树。加法树的输出被输入累加器，用于聚合相应的“部分激活值”。在“通信”阶段，与同一计算引擎中其他存储区的“部分激活值”一起，第 n 个计算引擎

5. 我们也以相同的方式对数据集 $\vec{A}$ 进行纵向分区。

6. 该数量可在编译时进行参数化配置，但需满足 FPGA 资源限制。然而，我们可以轻松地扩展 P4SGD，以支持存储在外部存储器（例如高带宽存储器（HBM））中的大型模型，而不会影响性能。

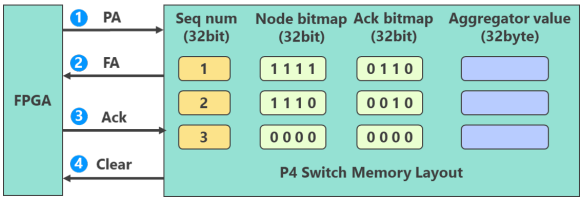


图 7：P4交换机与 FPGA 的交互　表 2：评估数据集

数据集	样本	特征	类别
gisette [27]	6,000	5,000	2
real_sim[7]	72,309	20,958	2
rcv1[7]	20,242	47,236	2
amazon_fashion[8]	200,000	332,710	5
avazu[7]	40,428,967	1,000,000	2

为当前包含 MB 个样本的微批次准备 $\overrightarrow{PA_{m\,n}}$。随后，第 m 个工作节点聚合其所有 N 个引擎的“部分激活值”，生成 $\overrightarrow{PA_m}$，并将其发送至图 3 中用于网内聚合的基于 P4 交换机的服务器。在“反向传播”阶段，我们还实例化了 64 个位串行乘法器，每个乘法器每个周期从 64 位“先进先出队列”中读取一个 1 位特征，并将其与对应的 32 位元素 $\overrightarrow{scale}[k]$ 相乘。因此，64 个位串行乘法器每个周期能够生成 2048 位梯度计算结果，以供相应的计算引擎进一步处理（第 4.1.2 小节）。

4.2 基于 P4 交换机的服务器实现

图 7 简要描述了 P4交换机与 FPGA 之间可靠传输的硬件实现。

我们的交换机数据平面使用 P4 语言实现。算法2 中的 agg、$agg\ count$、$agg\ bm$、$ack\ count$ 和 $ack\text{-}bm$ 均直接映射到 Tofino 寄存器阵列，如图 7 所示。在当前配置中，寄存器阵列的大小设为 64K（16 位索引），最多可支持 64K 个未完成的聚合操作。寄存器阵列分布在交换机流水线的 4 个阶段中，而流水线总共提供 12 个阶段。4 个阶段中任一阶段的资源消耗均限制在70.83% 可用 SRAM 的范围内。这为其他常规交换功能留出了充足的资源。我们利用 Tofino 数据包复制引擎实现向工作节点的多播。

5 评估

5.1 实验设置

工作负载。 我们在五个特征数量各异的数据集上进行实验，如表 2 所示。所有使用的数据集均可公开获取。

实验平台。我们在一个集群上开展实验，该集群包含两台网络交换机和八台机器。其中一台网络交换机为 Mellanox SN2700，配备 32 个 QSFP28 端口；另一台为 Wedge100BF-32X P4 可重构交换机，提供 32 个 QSFP28 端口和一个 20 MB 的数据包缓冲区。每台机器均配置了一颗 12 核/24 线程的 Intel Xeon(R) Silver 4214 CPU（2.2GHz）、一块 Xilinx

7. https://www.csie.ntu.edu.tw/∼cjlin/libsvmtools/datasets/ 8. 这是印度新闻数据集的一部分，该数据集可从 https://jmcauley.ucsd.edu/data/amazon/ 下载。

表 3：具有 8 个计算引擎的工作节点的资源消耗

硬件模块	LUTs	REGs	RAMs	DSPs	频率
PCI-Express	63K	98K	4.3Mb	0	250MHz
网络传输	10K	27K	3.5Mb	0	250MHz
高带宽存储器子系统	7K	42K	3.26Mb	0	450MHz
8 个引擎	188K	904K	152Mb	4096	250MHz
总利用率	304K (23%)	1.1M (42%)	165Mb (47.5%)	4096 (45%)	

Alveo U280 FPGA [18], 、一块 100Gb/s（支持远程直接内存访问）的 Mellanox MT27800 网卡和一块 NVIDIA A100 GPU （ 40 GB HBM2 内存，6912 个 CUDA 核心）。我们使用该平台进行所有实验。

P4SGD 实现。我们使用基于 P4 交换机的服务器和基于 FPGA 的工作节点实现了 P4SGD。表 3 展示了在 Xilinx Alveo U280 上使用 8 个引擎时 P4SGD 的资源消耗。总资源利用率约为 50%。P4SGD 支持灵活配置 1 到 8 个引擎。此外，我们还实现了一个数据并行训练系统，该系统同样利用 P4 交换机，在分布式 FPGA 之间进行网内聚合。该数据并行系统在一次迭代中的前向和反向传播结束后，聚合长度为 D 的梯度，而不是 B。采用 4 位精度，原因有二：1）精度降低时，执行时间会线性缩短；2）MLWeaving [24] 表明，低精度（高于 3 位）训练达到收敛所需的训练轮次数与 CPU 全精度方法相近。因此，我们选择 4 位精度，以缩短训练时间，同时不影响收敛速度。

GPU 基线。 GPU 基线（标记为“GPUSync”）在集群中的 GPU 上实现，采用同步式分布式线性模型随机梯度下降。“GPUSync”利用先进的 cuBLAS 库 [28] 高效实现前向和反向传播。具体而言，“GPUSync”使用函数 *cublasSgemm*，该函数可在单个 GPU 内自动并行化。通过 NVIDIA Nsight Systems，我们观察到*cublasSgemm* 在本实验中使用至少 512 个线程块，每个线程块含 128 个线程，以处理一个小批量的样本，这表明“GPUSync”已在一次 CUDA 调用中充分利用 GPU 算力。此外，我们还采用了一些 GPU 优化方法来加速训练：使用 CUDA Graphs [29] 减少内核调用开销，并使用支持 GPUDirect 的 RDMA+NCCL [30], [31] 减少节点间通信开销。“GPUSync”采用模型并行训练，显然比数据并行训练更快（第 5.3 小节）。然而，由于 CUDA 内核调用开销，“GPUSync”无法高效扩展；具体而言，每次训练迭代都需要启动三个 CUDA 内核：两个*cublasSgemm* 分别用于前向和反向传播，以及一个用于通信的 AllReduce。随着使用的 GPU 数量增加，每个内核可用的 GPU 计算周期会减少，而内核调用开销可能会主导总时间。

两个 CPU 基线方法。我们在分布式 CPU 上实现同步随机梯度下降算法，标记为“CPUSync”。我们在分布式 CPU 上采用以下优化方法：多核（12 核）、AVX2 指令（512 位）和基于远程直接内存访问的 openMPI（3.4.1 版本）库。“CPUSync”采用模型并行训练，在我们的实验中明显快于数据并行训练。另一个

CPU 基线是“SwitchML”，它采用与“CPUSync”相同的计算方法，但使用来自 SwitchML [16]，的通信方法，而非基于远程直接内存访问的 OpenMPI。

比较方法。我们的评估主要验证三个假设。第一，P4交换机-FPGA 网内聚合能够显著降低聚合延迟，从而有利于 GLM 的模型并行训练。第二，在 P4 交换机的帮助下，P4SGD 能够在分布式 FPGA 上实现近乎线性的扩展。第三，与分布式 GPU/CPU 上的同类方案相比，P4SGD 收敛更快且消耗更低。

5.2 聚合延迟比较

我们测量了 P4交换机-FPGA 网内聚合的延迟，该聚合用于在 FPGA 上进行分布式 GLM 训练（第 3.3 小节）。图 8 展示了 AllReduce 的延迟对比；每个工作节点上的数组 $\overrightarrow{PA}$ 包含 8 个 32 位元素，共有 8 个工作节点。我们有三个观察结果。

第一，P4SGD 的平均延迟可达 1.2μs，比“CPUSync”和“GPUSync”低一个数量级；这是因为交换机内聚合减少了额外的网络跳数，而我们的硬件实现降低了软件启动和同步开销。

其次，P4SGD 的延迟波动明显小于“CPUSync”和“GPUSync”，这证明了 P4SGD 在提供确定性延迟方面的优势。第三，SwitchML [16] 引入的延迟甚至比“CPUSync”和“GPUSync”更长，这是因为：1）SwitchML 利用影子副本机制延迟对已接收聚合数据包的确认，以获得更高的吞吐量；2）SwitchML 使用最小尺寸为 256B 的数据包，而其他方法采用 64B 的网络数据包。

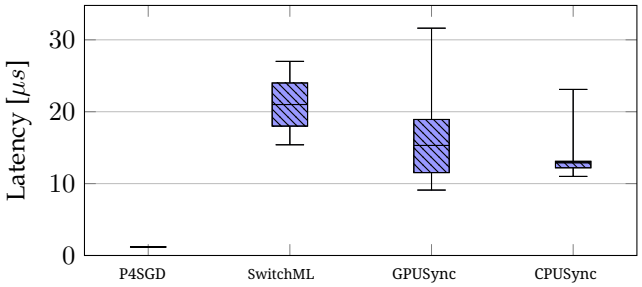
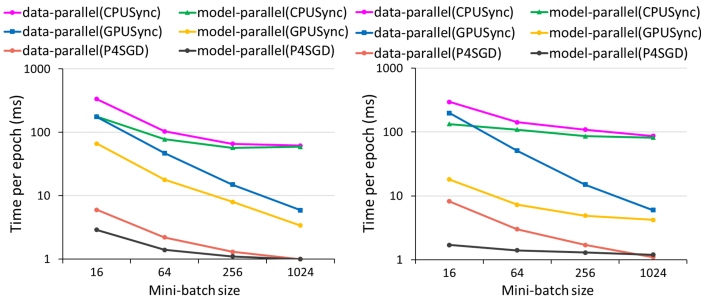


图 8：聚合延迟对比。须线表示第 1 和第 99 百分位数。

5.3 数据并行与模型并行的比较

在本节中，我们将数据并行和模型并行方法的每轮次耗时与另外两种基准方案“CPUSync”和“GPUSync”进行比较。所有实验中的工作节点数量均为 4。P4SGD 的计算引擎数量为 8。图 9 展示了在数据集 $rcv1$ 和$amazon\ \ fasion$上，不同小批量大小下的比较结果。我们有以下三个观察结果。首先，在大多数情况下，模型并行的每轮次耗时显著低于其对应的数据并行方法在娱乐主管



(a) $rcv1$(47K) (b) $amazon\ \ fasion$ (333K)

图 9：数据并行与模型并行的硬件效率比较，工作节点数为 4。

在相同平台上，这是因为模型并行引入的网络流量更少，尤其是在小批量大小较小时。尽管小批量大小较大时，数据并行需要更长的每轮时间，但要达到相同的收敛速度，所需的训练轮次显著更多，因此总体收敛加速比反而更小。其次，当 B 较小时，在相同平台上，模型并行相较于数据并行具有更高的加速比，因为 P4SGD 引入了硬件流水线并行以及采用 P4交换机-FPGA 的交换机内聚合，并具有超低延迟。例如，当 B 为 16 时，在 $amazon\ \ fasion$条件下，FPGA 上的模型并行比数据并行快 $4.8\times$；而当 B 为 1024 时，模型并行和数据并行的耗时大致相同。第三，特征数量越大，在相同平台上模型并行相较于数据并行的加速比越高。例如，当 B 为 16 时，在数据集 $rcv1$ 和$amazon\ \ fasion$上，在 $amazon\ \ fasion$条件下，P4SGD 分别比相应的 FPGA 数据并行实现快 $2\times$ 和 $4.8\times$。在后续实验中，我们始终采用模型并行。

5.4 硬件效率：吞吐量

我们从可实现吞吐量的角度考察 P4SGD 的硬件效率。首先，我们考察不同特性对 P4SGD 的影响。其次，我们将 P4SGD 与基于图形处理器和 CPU 的基线方法进行比较。在以下实验中，默认在每个工作节点内实例化 8 个计算引擎，以获得最佳性能。

5.4.1 P4SGD 的硬件特性

我们通常运行 200 个训练轮次，并计算平均吞吐量，以分析各项硬件特性的影响。

小批量大小的影响。我们研究了小批量大小对 P4SGD 的影响。我们采用 P4SGD-8-8 的实现。图 10 展示了在不同数据集上，与“B=16”的情况相比，P4SGD-8-8 在不同小批量大小下的吞吐量加速比。我们有两点观察。首先，较大的小批量大小会带来更高的加速比，因为较大的小批量大小能够让同一小批量中的微批次在前向/反向传播与通信之间实现重叠。其次，特征数量越多，增大小批量大小带来的加速比越低，因为即使 B 很小，P4SGD 也能够将通信时间与计算时间重叠；由于特征数量更多，计算时间在总计算时间中所占比例更高。

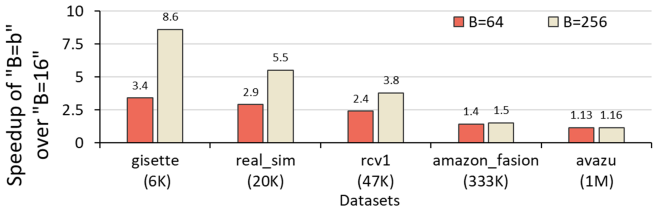


图10：在每个工作节点配备8个引擎、共8个工作节点的情况下，不同数据集上小批量大小对每轮时间的影响，B：小批量大小

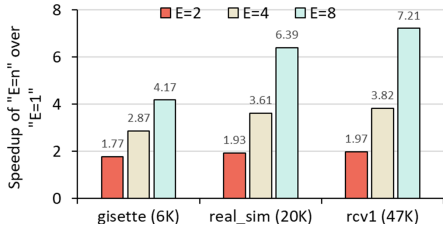


图11：在仅有1个工作节点的情况下，纵向扩展能力（以每轮时间计），E：引擎数量，小批量大小 =64，精度 = 4位

纵向扩展能力。我们研究了 P4SGD 通过实例化多个计算引擎来对 GLM 进行模型并行训练的纵向扩展能力。图 11 展示了在数据集 $gisette$、$real\ sim$ 和 $rcv1$ 上，与使用一个计算引擎的情况相比，使用不同数量计算引擎时的吞吐量比。我们有两点观察。首先，计算引擎数量越多，吞吐量越高，这是因为每个工作节点上的每个计算引擎所需处理的训练任务更少。其次，特征数量越多，增加计算引擎数量带来的吞吐量提升越大，因为特征数量越多，计算时间占总训练时间的比例越高，而引入更多计算引擎可以线性缩短计算时间。

横向扩展能力。我们考察了 P4SGD 使用多个工作节点对 GLM 进行模型并行训练时的横向扩展能力。图12显示了工作节点数量不同的情况下，相对于仅使用一个工作节点时的吞吐量比。我们有三点观察。第一，工作节点数量越多，吞吐量越高，因为每个工作节点承担的训练任务更少。第二，特征数量越多，增加工作节点数量带来的吞吐量提升越大，因为特征数量增加会提高计算时间在总训练时间中的占比，从而更容易抵消通信时间带来的负面影响。第三，当特征数量达到100万时，吞吐量会随着机器数量增加而近乎线性增长。这表明，当特征数量足够大时，P4SGD具有很强的横向扩展能力（$>1M$）。

5.4.2 与 CPU/GPU 基线的可扩展性比较

我们从每轮时间的角度，将 P4SGD 与另外三种基线方法“SwitchML”、“CPUSync”和“GPUSync”进行可扩展性比较。图 13 展示了不同工作节点数量和小批量大小下的平均每轮时间。我们有四点观察。

首先，“P4SGD”的速度显著快于其他三个对比方案，并且具有最高的可扩展性。这得益于

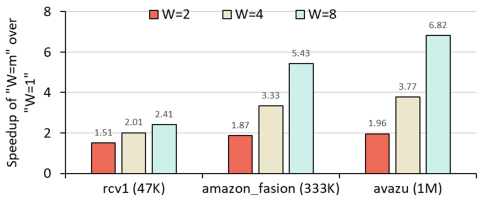


图12：P4SGD 在使用 8 个引擎时按每轮时间衡量的横向扩展能力，W：工作节点数量，小批量大小 = 16，以及精度 = 4位

其以延迟为中心的交换机内聚合协议和硬件流水线并行，从而最大限度地实现通信与前向/反向传播的重叠。

其次，当 B 相对较小时，“GPUSync”无法扩展，原因有二：1）由于内核调用开销严重（每次迭代需要调用三个内核），无法充分利用 GPU 算力；2）更多分布式图形处理器会加剧内核调用开销，从而抵消 CUDA 内核 *cublasSgemm* 计算时间缩短带来的收益，尤其是在维度相对较小时，如图 13a 所示。高性能的节点内 NVLink 可略微减轻通信开销，因为使用由 RDMA-GPUDirect 驱动的 NCCL 时，通信阶段仅占总训练时间的约 20%。

第三，“CPUSync”相对容易扩展，因为在分布式 CPU 上，计算时间占总训练时间的主要部分，而通信时间可以忽略不计。因此，随着工作节点数量增加，总训练时间会迅速缩短。

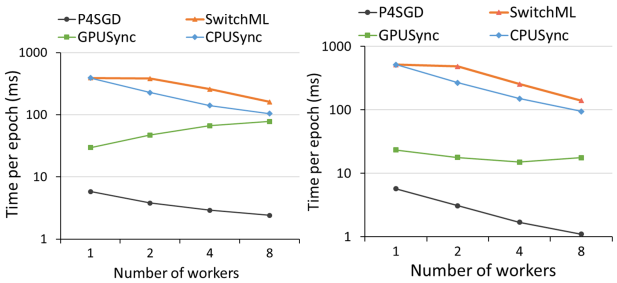
第四，“SwitchML”比“CPUSync”慢，扩展能力也不如“CPUSync”。这主要是因为“SwitchML”采用影子副本机制，延迟了对已接收聚合数据包的确认，因而具有最高的聚合延迟，如图 8 所示。实际上，“SwitchML”采用影子副本机制是为了大幅提升网内聚合的吞吐量，而非降低延迟。

5.5 统计效率：损失与训练轮次

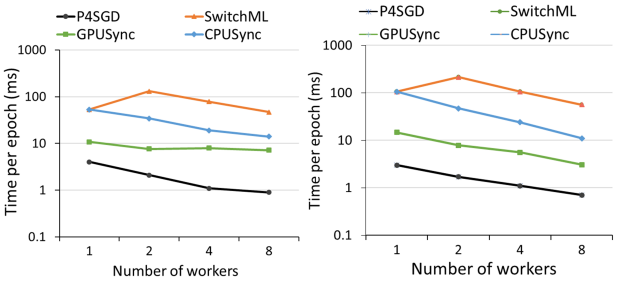
我们在数据集 $rcv1$ 和$avazu$上，将 P4SGD 的统计效率与“CPUSync”和“GPUSync”进行了比较。图 14 展示了训练轮次逐渐增加时的收敛趋势。所有方法的批量大小均为 64。我们观察到，所有方法都需要相同数量的训练轮次才能收敛到大致相同的损失，因为它们都是同步的。

5.6 端到端比较：损失与时间

我们将 P4SGD 的端到端性能与“CPUSync”和“GPUSync”进行比较，后两者均采用收敛时间最短的配置。图 15 展示了在数据集 $rcv1$ 和 $avazu$上的端到端收敛对比，以损失随时间的变化表示。我们有两点观察。首先，P4SGD 的收敛速度最高可比“GPUSync”快 6.5 倍，这表明网内聚合和微批次模型并行训练具有巨大潜力，能够在分布式 FPGA 上实现高效的模型并行训练。“GPUSync”收敛相对较慢，原因有二：1）每次迭代的通信开销较大；2）前向/反向传播与通信之间没有重叠

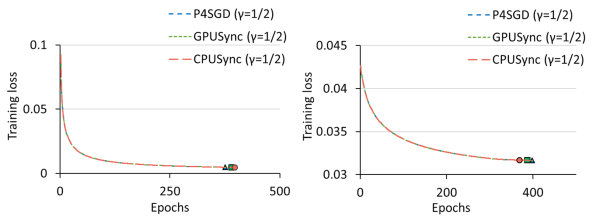


(a) $rcv1$(47K), B=16 (b) $amazon_fashion$ (333K), B=16



(c) $rcv1$(47K), B=256 (d) $amazon_fashion$ (333K), B=256

图13：P4SGD 与其他基线方法的硬件效率对比



(a) $rcv1$(47K) (b) $avazu$(1M)

图14：统计效率：训练损失与训练轮次的关系， γ：学习率，精度 = 4位

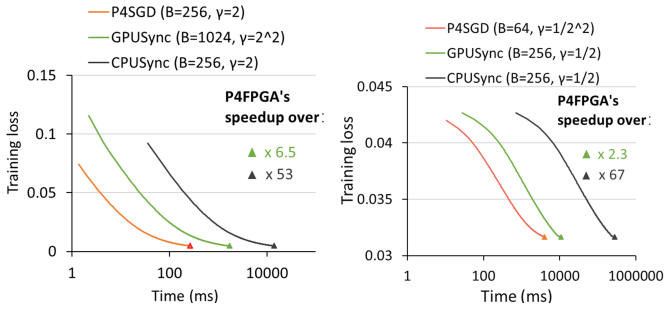
。其次，P4SGD 的收敛速度最高可比“CPUSync”快 67 倍，因为“CPUSync”的计算能力较弱，且集合通信延迟较长。

5.7 能耗

近年来，除了硬件效率和性能之外，能耗也已成为一项重要考量。P4SGD 在能耗方面具有显著优势。

为了测量 Xilinx Alveo U280 的功耗，我们使用 Alveo 卡管理解决方案子系统（CMS 子系统）。CMS 固件负责从卫星控制器设备收集 U280 的电压、电流、功耗及其他相关信息。 [32]“CPUSync”的功耗使用 lm sensors [33], 测量，这是一款开源应用程序，提供用于监测温度、电压和功耗的工具与驱动程序。“GPUSync”的功耗则使用 NVIDIA 系统管理接口（nvidia-smi） [34],测量，该接口可管理和监控 NVIDIA GPU 设备。

我们在端到端实验中评估了 P4SGD 与“CPUSync”和“GPUSync”的功耗。



(a) $rcv1$(47K) (b) $avazu$(1M)

图15：端到端收敛对比：训练损失与时间的关系，B：小批量大小， γ：学习率，精度为 = 4位 表4：能耗，所有方法的工作节点数量为8。

方法	数据集	时间（秒）	总功率(W)	能量(J)
P4SGD	rcv1	0.27	528	143
	avazu	4.12	528	2175
GPUSync	rcv1	1.76	920	1619
	avazu	10.9	920	10028
CPUSync	rcv1	14.4	496	7142
	avazu	128.25	496	63612

在第5.6节中。表4展示了能耗对比，其中不包括主机系统的功耗。结果表明，P4SGD的能效最高可达到“GPUSync”的11倍，并且比“CPUSync”高出50倍，体现了P4SGD在能耗方面的显著优势。

6 相关工作

分布式机器学习中的并行。 ColumnSGD [35] 提出了一种在多台 CPU 上采用模型并行的分布式随机梯度下降训练系统。随着机器数量增加，其迭代时间几乎保持不变。更糟的是，当机器数量从 10 台增加到 40 台时，ColumnSGD 的每次迭代时间增加了 1.3 倍。MegatronLM [12] 通过在节点内的 GPU 之间采用模型并行、在节点之间采用数据并行，高效训练数十亿参数的语言模型。Alpa [36]可自动为无法装入单个 GPU 的大型 深度学习模型的训练实现算子间和算子内并行。由于算子内并行的通信开销较高，Alpa 建议将其用于通过 NVLink 等高带宽通信链路连接的加速器，并建议将算子间并行用于通过相对低带宽网络连接的分布式加速器。GSPMD [37] 会根据用户注释自动完成每个张量的分片；用户可以结合数据并行、模型并行和流水线并行等方式。然而，GSPMD 不会在流水线并行中对单个算子或张量进行分区，而是将训练图划分为多个阶段，并在不同设备上运行这些阶段。相比之下，P4SGD 致力于借助可编程交换机，高效地利用算子内并行（本文中的模型并行），例如实现强扩展，从而在 FPGA 等分布式加速器之间训练全连接层（即 GLM 模型）。

分布式机器学习中的流水线并行。 以往的流水线并行方法，例如 GPipe [38], HetPipe [39], TeraPipe [14]

将模型的不同层分配给不同的 GPU 进行训练，从而实现大模型训练；因此，它与面向模型并行的 P4SGD 正交。

FPGA 加速的机器学习系统。 P4SGD与MLWeaving [24]，最为接近，后者是一个适用于任意精度学习的通用系统。MLWeaving 的硬件效率与使用 1 个计算引擎和 1 个工作节点的 P4SGD 相当。已有工作 [40]–[53]利用 FPGA 实现高性能，但并未用于模型并行训练。Brainwave [45]通过在多个 FPGA 上采用模型并行来加速深度神经网络推理。先前的工作 [53] 提出了一个用于加速分布式支持向量机训练的多 FPGA 系统，但所有 FPGA 都位于同一台主机上，并通过 PCIe 相互通信。因此，其在单台主机中的可扩展性受到限制。

网内聚合。 当前研究 ATP [17]、SwitchML [16] 和 [54] 利用可编程网络交换机执行数据并行训练中的梯度聚合。此外，它们采用影子副本机制来优化吞吐量。相比之下，P4SGD针对超低网络内聚合延迟进行优化，以支持分布式 FPGA 之间的模型并行训练。SHARP [55]将集合操作处理卸载到 InfiniBand 网络中。SHARP采用分层聚合树架构，以支持数百台主机上的通用聚合操作。Herring [56] 是一个具有可扩展性的、基于参数服务器的分布式数据并行训练库，采用平衡的融合缓冲区来解决服务器发送和接收数据不均衡的问题。Herring主要通过提高聚合操作的吞吐量来实现数据并行分布式训练，而P4SGD主要通过降低聚合操作的延迟来实现模型并行分布式训练。

7 结论

我们提出了 P4SGD，这是一种模型并行分布式训练系统，可在训练广义线性模型时实现强扩展。P4SGD采用微批次硬件流水线并行训练，在工作节点内重叠前向/反向传播与通信。同时，我们提出了一种以延迟为中心的交换机内聚合协议，以降低分布式 FPGA 之间的通信开销。我们在多块 FPGA 和一台 P4交换机上实现了 P4SGD 原型。实验结果表明，P4SGD 的收敛速度最高可比其 GPU 对应方案快 9.3 倍。我们将开源 P4SGD，以促进社区发展。

局限性与未来工作。主要局限在于，P4SGD 将模型部署在 FPGA 的片上存储器中，因此受片上存储器容量限制，模型大小也受到限制。当前实现支持可参数化的模型大小，但上限为 2M。不过，我们可以轻松扩展 P4SGD，通过将模型存储在外部存储器（例如高带宽存储器）中来支持更大的模型。为此，大模型的实现需要更高的内存带宽才能实现线速硬件处理。幸运的是，当前的 P4SGD 仅使用了 25% 的高带宽存储器带宽，因此大部分带宽可供计算引擎访问外部存储器中的大模型。我们认为 P4SGD 的处理速度不会因此降低。相关实现将留待未来工作完成。

另一项未来工作是将 P4SGD 扩展至采用 FPGA-GPU 协同处理方法进行深度神经网络训练。具体而言，我们通过数据并行在 GPU 上训练计算密集型层，例如卷积层，因为这些层的每个权重对应的计算量较大；而通过模型并行在 P4SGD 上训练全连接层，因为这些层的模型大小可能较大，但每个权重对应的计算量较小。

未来工作的第三项是在 225兆赫兹频率下运行 P4SGD，使其能够处理来自单个高带宽内存通道的 512位数据，从而使当前设计支持的引擎数量翻倍。

致谢。 本研究得到以下项目资助：中国国家重点研发计划（项目编号：2022ZD0119301）、浙江省科学技术计划（2022C01044）、中央高校基本科研业务费专项资金（226-2022-00151）、浙江大学上海高等研究院星夜科学基金（SN-ZJU-SIAS-0010）、浙江省角膜疾病研究重点实验室，以及阿里巴巴集团通过阿里巴巴创新研究（AIR）计划提供的研究经费。

参考文献

[1] S. Rendle，《因子分解机》，发表于国际数据挖掘会议，2010 年。[2] A. Elgohary、M. Boehm、P. J. Haas、F. R. Reiss 和 B. Reinwald，《面向大规模机器学习的压缩线性代数》，发表于《数据库领域国际期刊》，2016 年。[3] H. Liu、K. Wang、Z. Chen 和 K. E. Jordan，《并行分布式系统上用于高度异构油藏模拟的高效多阶段预条件器》，发表于*SPE*储层模拟研讨会，2015 年。[4] T. Zheng、G. Chen、X. Wang、C. Chen、X. Wang 和 S. Luo，《实时智能大数据处理：技术、平台与应用》，发表于《中国科学：信息科学》，2019 年。[5] P. Pop、M. L. Raagaard、S. S. Craciunas 和 W. Steiner，《利用 IEEE 时间敏感网络优化信息物理分布式系统设计》，发表于《*IET*信息物理系统：理论与应用》，2016 年。[6] L. Oden、B. Klenk 和 H. Fr¨oning，《分布式图形处理器上的节能集合归约和全归约操作》，发表于*CC- GRID*，2014 年。[7] J. Lee、I. Hwang、S. Shah 和 M. Cho，《FlexReduce：面向非对称网络拓扑上分布式深度学习的灵活全归约》，发表于设计自动化会议，2020 年。[8] M. Li、D. G. Andersen、J. W. Park、A. J. Smola、A. Ahmed、V. Josi- fovski、J. Long、E. J. Shekita 和 B.-Y. Su，《利用参数服务器扩展分布式机器学习》，发表于操作系统设计与实现研讨会，2014 年。[9] Q. Ho、J. Cipar、H. Cui、S. Lee、J. K. Kim、P. B. Gibbons、G. A. Gibson、G. Ganger 和 E. P. Xing，《通过陈旧同步并行参数服务器实现更有效的分布式机器学习》，发表于神经信息处理系统大会，2013 年。[10] H. Cui、H. Zhang、G. R. Ganger、P. B. Gibbons 和 E. P. Xing，《Geeps：利用 GPU 专用参数服务器在分布式图形处理器上实现可扩展深度学习》，发表于欧洲计算机系统会议，2016 年。[11] A. Krizhevsky，《并行化卷积神经网络的一个奇妙技巧》，发表于计算研究资料库，2014 年。[12] M. Shoeybi、M. Patwary、R. Puri、P. LeGresley、J. Casper 和 B. Catanzaro，《Megatron-LM：使用模型并行训练数十亿参数的语言模型》，发表于*arXiv* 预印本 *arXiv:1909.08053*，2020 年。[13] Z. Jia、M. Zaharia 和 A. Aiken，《超越深度神经网络的数据并行与模型并行》，发表于*arXiv* 预印本 *arXiv:1807 05358*，2018。。[14] Z. Li、S. Zhuang、S. Guo、D. Zhuo、H. Zhang、D. Song 和 I. Stoica，《TeraPipe：用于训练大规模语言模型的令牌级流水线并行》，发表于计算研究资料库，2021 年。[15]微软，《ZeRO 与 DeepSpeed：全新系统优化助力训练参数量超过 1000 亿的模型》，https://www.microsoft.com/en-us/research/blog/zero-deepspeed-new-system-optimizations-enable-training-models-with-over-100-billion-parameters/。

[16] A. Sapio、M. Canini、C.-Y. Ho、J. Nelson、P. Kalnis、C. Kim、A. Krishnamurthy、M. Moshref、D. R. Ports 和 P. Richt´arik，《利用网内聚合扩展分布式机器学习》，arXiv 预印本 arXiv:1903.06701，2019 年。[17] C. Lao、Y. Le、K. Mahajan、Y. Chen、W. Wu、A. Akella 和 M. M. Swift，《ATP：用于多租户学习的网内聚合》，发表于 NSDI，2021 年。[18] Xilinx，《Alveo U280 数据中心加速卡数据手册》，https://www.xilinx.com/support/documentation/data sheets/ds963-u280.pdf，2020 年。[19] Edge-core，《Wedge 100BF-32X R07》，https://www.edge-core.co m/ upload/images/Wedge100BF-32X 65X DS R07 20200731. pdf，2020 年。[20]Intel，《Intel Tofino 可编程以太网交换机 ASIC》，https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html。[21] Intel，《Intel Tofino 2 P4 可编程以太网交换机 ASIC》，https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-2-series.html。[22] P. Bressana、N. Zilberman、D. Vucinic 和 R. Soul´e，《以网络计算换取延迟》，发表于 NAI，2020 年。[23] Q. Wang、Y. Lu、E. Xu、J. Li、Y. Chen 和 J. Shu，《Concordia：具有网内缓存一致性的分布式共享内存》，发表于 FAST，2021 年。[24] Z. Wang、K. Kara、H. Zhang、G. Alonso、O. Mutlu 和 C. Zhang，《利用 MLWeaving 加速广义线性模型：适用于任意精度学习的通用系统》，发表于《VLDB 会议录》，2019 年。[25] C. Eckert、X. Wang、J. Wang、A. Subramaniyan、R. Iyer、D. Sylvester、D. Blaaauw 和 R. Das，《神经缓存：深度神经网络的缓存内位串行加速》，发表于 2018 年 ACM/IEEE 第 45 届计算机体系结构国际研讨会（ISCA）。IEEE，2018 年，第 383–396 页。[26] Y. Umuroglu、L. Rasnayake 和 M. Sj¨alander，《Bismo：面向可重构计算的可扩展位串行矩阵乘法覆盖层》，发表于 2018 年第 28 届现场可编程逻辑与应用国际会议（FPL）。IEEE，2018 年，第 307–3077 页。[27] C.-C. Chang 和 C.-J. Lin，《LIBSVM：支持向量机库》，发表于 TIST，2011 年。[28] NVIDIA，《cuBLAS》，2010 年。[29] Alan Gray，《CUDA Graphs 入门》，https://developer.nvidia.com/blog/cuda-graphs/，2019 年。[30] NVIDIA，《利用 NVIDIA Magum IO GPUDirect Storage 加速 GPU 与存储之间的通信》，https://nvdam.widen.net/s/k8vrp9xkft/tech-overview-magnum-io-1790750-r5-web，2021 年。[31] NVIDIA，《NVIDIA 集合通信库（NCCL）文档》，https://docs.nvidia.com/deeplearning/nccl/ user-guide/docs/index.html，2020 年。[32] Xillinx，《Alveo 卡管理解决方案子系统产品指南（PG348）》，https://docs.xilinx.com/r/en-US/pg348-cms-subsystem/Introduction。[33] Free Software Foundation, Inc.，《lm-sensors》，https://github.com /lm-sensors/lm-sensors。[34] NVIDIA，《NVIDIA 系统管理接口》，https://developer.nvidia.com/nvidia-system-management-interface。[35] Z. Zhang、W. Wu、J. Jiang、L. Yu、B. Cui 和 C. Zhang，《ColumnSGD：面向分布式随机梯度下降的面向列的框架》，发表于 ICDE，2020 年。[36] L. Zheng、Z. Li、H. Zhang、Y. Zhuang、Z. Chen、Y. Huang、Y. Wang、Y. Xu、D. Zhuo、J. E. Gonzalez 等，《Alpa：自动化分布式深度学习的算子间与算子内并行》，arXiv 预印本 arXiv:2201.12023，2022 年。[37] Y. Xu、H. Lee、D. Chen、B. Hechtman、Y. Huang、R. Joshi、M. Krikun、D. Lepikhin、A. Ly、M. Maggioni 等，《GSPMD：用于机器学习计算图的通用可扩展并行化》，arXiv 预印本 arXiv:2105.04663，2021 年。[38] Y. Huang、Y. Cheng、A. Bapna、O. Firat、D. Chen、M. Chen、H. Lee、J. Ngiam、Q. V. Le、Y. Wu 和 z. Chen，《GPipe：利用流水线并行高效训练巨型神经网络》，发表于 NeurIPS，2019 年。[39] J. H. Park、G. Yun、M. Y. Chang、N. T. Nguyen、S. Lee、J. Choi、S. H. Noh 和 Y.-r. Choi，《Hetpipe：通过整合流水线模型并行与数据并行，在（性能较弱的）异构 GPU 集群上实现大型深度神经网络训练》，发表于 ATC，2020 年。[40] A. Boutros、S. Yazdanshenas 和 V. Betz，《拥抱多样性：面向低精度深度学习的增强型数字信号处理器模块》，发表于*FPL*，2018 年。[41] S. Cadambi、I. Durdanovic、V. Jakkula、M. Sankaradass、E. Cosatto、S. Chakradhar 和 H. P. Graf，《用于支持向量机的大规模并行 FPGA 协处理器》，发表于*FCCM*，2009 年。[42] H. Sharma、J. Park、D. Mahajan、E. Amaro、J. K. Kim、C. Shao、A. Mishra 和 H. Esmaeilzadeh，《从高级深度神经网络模型到 FPGA》，发表于*MICRO*，2016 年。[43] A. M. Caulfield、E. S. Chung、A. Putnam、H. Angepat、J. Fowers、M. Haselman、S. Heil、M. Humphrey、P. Kaur、J.-Y. Kim 等，《云规模加速架构》，发表于*MICRO*，2016 年。[44] G. R. Chiu、A. C. Ling、D. Capalija、A. Bitar 和 M. S. Abdelfattah，《灵活性：用于深度学习加速的 FPGA 与 CAD》，发表于*ISPD*，2018 年。[45] E. Chung、J. Fowers、K. Ovtcharov、M. Papamichael、A. Caulfield、T. Massengill、M. Liu、D. Lo、S. Alkalay、M. Haselman 等，《借助 Project Brainwave 在数据中心规模上实时提供 DNN 服务》，*MICRO*，2018 年。[46] C. De Sa、M. Feldman、C. R´e 和 K. Olukotun，《理解并优化异步低精度随机梯度下降》，发表于*ISCA*，2017 年。[47] J. Fowers、K. Ovtcharov、M. Papamichael、T. Massengill、M. Liu、D. Lo、S. Alkalay、M. Haselman、L. Adams、M. Ghandi 等，《面向实时 AI 的可配置云规模 DNN 处理器》，发表于*ISCA*，2018 年。[48] K. Kara、D. Alistarh、G. Alonso、O. Mutlu 和 C. Zhang，《FPGA 加速的稠密线性机器学习：精度与收敛性的权衡》，发表于*FCCM*，2017 年。[49] Z. Li、C. Ding、S. Wang、W. Wen、Y. Zhuo、C. Liu、Q. Qiu、W. Xu、X. Lin、X. Qian 等，《E-RNN：面向 FPGA 高效循环神经网络的设计优化》，发表于*HPCA*，2019 年。[50] D. Mahajan、J. K. Kim、J. Sacks、A. Ardalan、A. Kumar 和 H. Esmaeilzadeh，《RDBMS 内高级分析的硬件加速》，*arXiv* 预印本 *arXiv:1801.06027*，2018 年。[51] D. Mahajan、J. Park、E. Amaro、H. Sharma、A. Yazdanbakhsh、J. K. Kim 和 H. Esmaeilzadeh，《Tabla：用于加速统计机器学习的统一模板框架》，发表于*HPCA*，2016 年。[52] T. Moreau、M. Wyse、J. Nelson、A. Sampson、H. Esmaeilzadeh、L. Ceze 和 M. Oskin，《SNNAP：通过神经加速在可编程 SoC 上进行近似计算》，发表于*HPCA*，2015 年。[53] J. Dass、Y. Narawane、R. N. Mahapatra 和 V. Sarin，《多 FPGA 系统上的支持向量机分布式训练》，*TC*，2020 年。[54] L. Luo、J. Nelson、L. Ceze、A. Phanishayee 和 A. Krishnamurthy，《Parameter Hub：面向分布式深度神经网络训练的机架级参数服务器》，发表于*SoCC*，2018 年。[55] R. L. Graham、D. Bureddy、P. Lui、H. Rosenstock、G. Shainer、G. Bloch、D. Goldenerg、M. Dubman、S. Kotchubievsky、V. Koush-nir 等，《可扩展分层聚合协议（SHArP）：高效数据归约的硬件架构》，发表于*COMHPC*，2016 年。[56] I. Thangakrishnan、D. Cavdar、C. Karakus、P. Ghai、Y. Selivonchyk 和 C. Pruce，《Herring：重新思考云规模下的参数服务器》，发表于*SC*，2020 年。



Hongjing Huang 目前是中国浙江大学的工程博士生。此前，他在浙江大学获得硕士和学士学位。他的研究兴趣包括分布式机器学习、智能网卡等。



Yingtao Li 目前是中国浙江大学的工程博士生。他的研究兴趣包括网内计算、可编程交换机应用等。



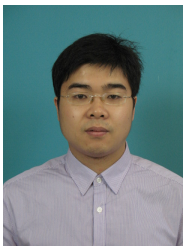
Jialin Li 于2019年获得华盛顿大学哲学博士学位。Li目前是新加坡国立大学计算机学院的助理教授。他的研究兴趣包括与数据中心网络、数据平面操作系统以及用于可编程网络硬件的系统软件协同设计分布式系统。



Jie Sun 目前是中国浙江大学的哲学博士生。此前，他获得了浙江大学学士学位。他的研究兴趣包括图神经网络、机器学习系统等。



Xueying Zhu 目前是中国浙江大学的哲学博士生。此前，她获得了浙江大学学士学位。她的研究兴趣包括网内计算、智能网卡等。



Zeke Wang 于2011年获得中国浙江大学哲学博士学位。他是中国浙江大学人工智能协同创新中心计算机科学系的研究教授。目前，他的研究兴趣主要集中于利用异构设备构建机器学习系统，例如智能网卡和智能交换机。



Jie Zhang 目前是中国浙江大学的哲学博士生。此前，他获得了浙江大学学士学位。他的研究兴趣包括云存储、智能网卡等。



Liang Luo 于2020年获得华盛顿大学哲学博士学位。他的研究重点是提高分布式训练的效率。