doi:10.11887/j.cn.202405022 http://journal.nudt.edu.cn

面向大规模异构计算平台的MiniGo高效训练方法

李荣春,贺周雨*,乔 鹏,姜晶菲,窦 勇,李东升
(国防科技大学 并行与分布计算全国重点实验室,湖南 长沙 410073)

摘 要:提出一种适用于大规模异构计算平台训练MiniGo智能体的高效多级并行训练方法,包括节点间任务级并行、中央处理器-数字信号处理器(central processing unit-digital signal processor, CPU-DSP)异构并行、DSP核内并行。实现了高效的输入/输出部署,消除网络通信瓶颈。提出了面向CPU-DSP共享内存结构的异构计算内存管理,减少异构设备间的数据搬运。实现了共享内存编程优化,并利用DSP实现密集卷积计算算子加速优化。结果表明,与16核CPU计算相比,单核DSP算子加速最大加速比达16.44;该方法实现计算节点规模从1 067扩展至4 139,得到达到给定终止条件所需时间从43.02 h降至16.05 h,可扩展效率为69.1%。评估表明,该方法能够实现MiniGo在大规模异构计算平台的高效并行训练。

关键词:MiniGo;大规模异构计算平台;数字信号处理器
中图分类号:TP39 **文献标志码**:A **文章编号**:1001-2486(2024)05-209-10

High efficient training method of MiniGo on large-scale heterogeneous computing platform

LI Rongchun, HE Zhouyu, QIAO Peng, JIANG Jingfei, DOU Yong, LI Dongsheng*
(National Key Laboratory of Parallel and Distributed Computing, National University of Defense Technology, Changsha 410073, China)

Abstract: An efficient multi-level parallel training method suitable for training MiniGo agents on large-scale heterogeneous computing platforms was proposed, including task level parallelism between nodes, CPU-DSP(central processing unit-digital signal process) heterogeneous parallelism and DSP core parallelism. Efficient input/output deployment and eliminated the bottleneck of network communication were realized. A heterogeneous computing memory management oriented to CPU-DSP shared memory structure was proposed to reduce the data handling between heterogeneous devices. Shared memory programming optimization was realized, and the dense convolution calculation operator acceleration optimization was realized by DSP. Results show that compared with 16 core CPU calculation, the maximum acceleration ratio of single core DSP operator acceleration is 16.44. In this method, the scale of computing nodes is expanded from 1 067 to 4 139, the time required to reach the given termination condition is reduced from 43.02 h to 16.05 h, and the expansion efficiency is 69.1%. Evaluation shows that this method can realize the efficient parallel training of MiniGo on large-scale heterogeneous computing platforms.

Keywords: MiniGo; large-scale heterogeneous computing platform; DSP

强化学习(reinforcement learning,RL)是处理决策问题的有效方法之一。强化学习中,智能体通过策略做出动作,在和环境的交互中收集样本,最大化长期回报,从而逐渐适应环境。借助深度神经网络(deep neural networks, DNNs)技术,AlphaGo[1]、AlphaStar[2]等深度强化学习(deep reinforcement learning,DRL)智能体在围棋、星际争霸等比赛中打败了人类职业选手。DRL成为当前机器学习领域的研究热点之一,被广泛应用于机器人控制、资源调度、自动驾驶等领域[3-8]。

为了优化提升智能体的决策能力,DRL方法需要收集大量任务相关的样本数据来训练。以Atari中的深度Q学习网络[9](deep Q-learning network, DQN)训练为例,为了复现论文中智能

收稿日期:2022-06-27
基金项目:国家自然科学基金资助项目(61902415)
作者简介:李荣春(1985—),男,安徽无为人,研究员,博士,硕士生导师,E-mail:rongchunli@nudt.edu.cn
***通信作者**:贺周雨(1995—),女,贵州兴义人,硕士,E-mail:he535040@163.com
引用格式:李荣春,贺周雨,乔鹏,等. 面向大规模异构计算平台的MiniGo高效训练方法[J]. 国防科技大学学报, 2024, 46(5): 209-218.
Citation: LI R C, HE Z Y, QIAO P, et al. High efficient training method of MiniGo on large-scale heterogeneous computing platform[J]. Journal of National University of Defense Technology, 2024, 46(5): 209-218.

体的水平,需要智能体与游戏交互千万步。同时,巨大的状态空间、动作空间和长期稀疏的奖励导致DRL智能体训练对算力需求高。OpenAI Five[10]使用了256块P100显卡,持续训练了10个月。正是由于DRL算法计算模式复杂、对算力需求高的特点,使得其适合作为大规模计算平台的系统性能评测应用。如何根据系统结构特性,提高训练速度和智能体能力成为研究热点之一。

MLPerf[11]中强化学习测试程序选型为MiniGo。MiniGo涉及对弈仿真模拟、深度神经网络的推理以及训练,计算过程复杂,需要面向系统结构进行定制化适配。当前,MiniGo在超算平台适配研究中通常采用通用中央处理器(central processing unit, CPU)或者图形处理器(graphic processing unit, GPU)的硬件平台,针对第三方芯片的适配处于空白。同时,当前MiniGo超算适配优化工作中,存在没有对算子进行定制化汇编优化、冗余操作导致读写频繁、计算节点间通信开销大、未针对异构设备特点进行针对性并行优化等问题,出现训练效率低、可扩展效率低的现象。针对以上问题,本文提出了一种高效的多层并行策略,实现MiniGo训练中计算任务的高效并行。结合原型系统结构特点和MiniGo训练计算模式,包含节点间任务级并行、中央处理器-数字信号处理器(central processing unit-digital signal processor, CPU-DSP)异构并行、DSP核内并行。结合MiniGo训练中对文件系统的操作和原型系统共享文件系统的特性,实现了一种高效输入/输出(input/output, I/O)模块。通过优化样本文件、模型文件的读写特性和改写训练中数据的读写行为,有效减少了大规模分布式计算的I/O开销。面向CPU-DSP共享内存结构的异构计算内存管理,有效降低CPU-DSP异构计算中的数据搬运,实现CPU和DSP之间的流水并行优化。同时将算子计算卸载到DSP,CPU负责进行蒙特卡罗树搜索(Mento Carlo tree search, MCTS)、网络通信、调度等任务。针对各算子计算-访存特性,设计了MiniGo应用中典型算子并行优化方案,实现了典型算子的高效计算。

1 相关工作

1.1 相关应用

2013年DeepMind提出DQN,融合深度学习与传统强化学习Q-learning方法,在以Atari为代表的决策类应用中取得了突破性进步。随后DRL在围棋、即时战略游戏等领域取得里程碑式的突破。MLPerf选择MiniGo为强化学习基准测试应用。

MiniGo是根据AlphaGo Zero论文编写的开源围棋智能体训练代码。MLPerf封闭模型分区的强化学习基准测试应用MiniGo提供的是单节点执行版本,其流程如算法1伪代码所示。主要分为两个阶段:训练模型和评估模型。阶段1又可分为训练和自我博弈。从当前模型前5个模型产生样本中随机采样作为数据集。迭代1个epoch得到新模型。新模型进行自我博弈,得到对应的样本集,加入下一次迭代。当最新模型产生指定对局数后,开启下一次训练。重复阶段1,得到100个模型。进入阶段2进行模型评估。训练得到模型依次与目标模型对弈,记录胜率。当胜率超过50%,认为当前模型足够智能,达到训练目标,输出训练该模型所需时间。

算法1 单节点版本的MiniGo算法

Alg. 1 MiniGo algorithm of single node version

输入:训练模型M,读模型样本窗口数W,自我博弈阶段对弈局数N,模型自我博弈产生样本S,模型评估对局数E,目标模型T

输出:达到50%胜率模型所需训练时间t

根据模型M初始化参数
for $i=1:100$ **do**
  从最近W个模型产生的样本$S_\{i-1\},\cdots,S_\{i-W\}$中随机采样,作为数据集
  迭代1个epoch训练得到新模型M_i
  新模型M_i进行自我博弈,产生样本S_i
  监控S_i数量,达到N局时,启动下一次训练;否则,等待
end for
for $i=1:100$ **do**
  模型S_i与目标模型T对弈E局,统计胜率
  胜率超过50%,输出t,中止
end for

1.2 深度强化学习方法、分布式和并行化

DRL需要收集大量智能体和环境交互的数据。利用样本训练来提升智能体的决策能力。DRL相关工作中除了训练算法方面的改进外,还有围绕高效利用计算集群进行的研究。下面着重对高效利用计算集群方面的相关研究工作进行梳理。

Ong 等[12]使用 DistBelief[13]实现了 DQN 的多节点异步训练。2015 年，DeepMind 构建了第一个用于深度强化学习的大规模分布架构 Gorila[14]。Mnih 等提出的 A3C[15]算法针对 Gorila 框架的限制，使用了 1 台机器的多个 CPU 线程作为多个异步 actors-learners，挖掘并行探索样本多样性对加快训练收敛速度的影响。Ape-X[16]设计了分布式优先级经验回放策略，使智能体能够从比以前更多数量级的数据中有效地学习。IMPALA[17]提出了一种被称为 V-trace 的非策略校正方法，该方法保证了智能体在高吞吐量下的稳定学习。IMPALA 框架也实现了扩展到数百台机器的分布式学习。除了结合具体 DRL 应用和算法的框架，研究人员也关注分布式训练框架。例如 RLlib[18]，一种基于 Ray[19]开发的强化学习开发工具集。

1.3　MLPerf 中 MiniGo 提交情况分析

MLPerf 是一套测量机器学习性能的基准，在工业界和学界得到广泛的认可。强化学习应用分区的基准测试应用是 MiniGo。MiniGo 是根据 AlphaGo Zero 论文编写的开源围棋智能体训练代码。已向 MLPerf 榜单提交 MiniGo 项目实验结果的机构有：NVIDIA 和 Intel。对它们提交的代码和实验结果进行分析。NVIDIA 以 dgxa100_ngc20.06 机器作为 1 个计算节点。每台 dgxa100_ngc20.06 拥有 2 个 AMD EPYC 7742 CPU；8 个 A100-SXM4-40 GB GPU。计算节点规模分别为 1 个节点、2 个节点、32 个节点和 224 个节点。分析 NVIDIA 提交的代码，使用 OPENMPI 进行通信，调用 Horovod 框架[20]实现分布式训练。优化的重点是 DNNs 的推理训练，比如推理时数据格式为 BF16 等。分析 NVIDIA 提交的实验结果，单节点得到合格智能体需要训练18 770 s，节点规模增加至 224 个时需要训练1 100 s，可扩展效率约 7.8%。达到终止条件，智能体所需迭代次数随着节点数增加而增加，从 42 次迭代增加到 65 次迭代。随着节点数增大，dgxa100_ngc20.06 系统将面临的问题是：①迭代次数增大；②网络通信成为性能瓶颈。Intel 提交的结果和优化代码也有类似的倾向。优化的重点是采用 INT8 的推理优化。

2　高效训练方法

2.1　系统架构

本文方法所依托的计算平台为如图 1 所示。原型系统硬件平台由高速网络、并行文件系统、异构计算芯片等构成。原型系统软件平台为计算任务池提供任务调度、任务分配、数据管理和负载均衡服务。原型系统的操作系统内核版本为 Ubuntu 19.04，编译环境为 gcc8.3.0，并行文件系统为 Lustre，信息传递服务由 TH Express 提供。

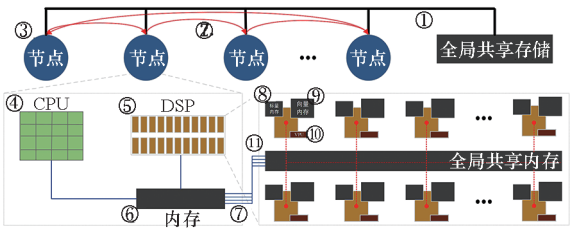


图 1　大规模异构计算平台体系结构

Fig. 1　System architecture of large-scale heterogeneous computing platform

原型系统中，节点间通信使用的是自建高速网络，分为配置网络和数据网络，如图 1 中双实线②所示，通信带宽 100 Gbit/s。节点间共享存储使用的是自研高速网络，如图 1 中单实线①所示，通信带宽 100 Gbit/s。

图 1 中，每个计算节点③由 CPU④和 DSP⑤构成 CPU-DSP 异构计算节点。CPU 和 DSP 簇之间共享双倍数据速率（double data rate，DDR）内存空间⑥。DDR 和 DSP 端数据之间有多个直接存储访问（direct memory access，DMA）通道⑦，实现 DDR 和 DSP 端之间的高速数据访问，读写带宽42.6 GB/s。每个计算节点包含 1 个 16 核 ARMv8 CPU 和 1 个 DSP 簇。16 核 CPU 是裁剪版的 Phytium FT-2000plus Processor。CPU 核可以访问 DSP 所有共享资源，DSP 核只能访问 DSP 核内私有的资源和 DSP 簇内共享的资源。DSP 簇包含 24 个 DSP 核，1 个 6 MB 的片上全局共享内存（global shared memory，GSM）⑪。DSP 单核内拥有 1 个 64 KB 的私有标量内存（scalar memory，SM）⑧、1 个 768 KB 的私有向量内存（array memory，AM）⑨、一个向量处理单元（vector processing unit，VPU）⑩。VPU 由 16 个向量处理部件（vector processing element，VPE）组成，可以实现高吞吐单精度浮点计算。单核 DSP 在 1.8 GHz下处理 FP32 的峰值性能是 345.6 GFlops。

2.2　高效多级并行策略

MiniGo 应用计算复杂，主要分为训练和自我博弈，如算法 1 所示。训练和自我博弈任务互相依赖，交替进行。为了充分发挥原型系统中计算节点多、加速卡计算速度快等优势，训练任务可按

照多节点间数据并行、节点内 CPU-DSP 异构流水并行、DSP 核间并行以及 DSP 指令集并行进行。同理，自我博弈任务也按照这三级并行对任务进行映射。

本文设计了一种高效的多级并行策略，如图 2所示。该策略包括节点并行、节点内 CPU-DSP 异构流水并行和 DSP 核间并行以及 DSP 指令集并行。

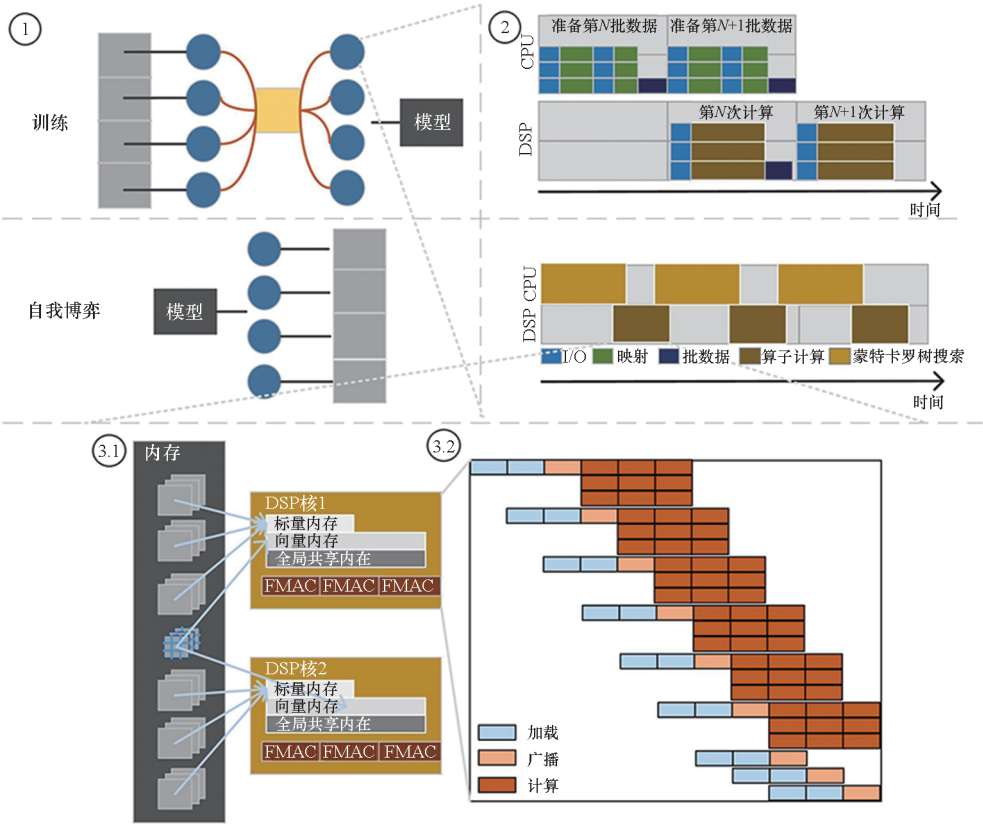


图 2 高效多级并行策略示意图

Fig. 2 Schematic diagram of efficient multi-level parallel strategy

2.3 节点并行

算法 1 的训练任务、自我博弈任务的计算任务按照上述三级并行策略映射。使用原型系统 $N_{\text{train}}+N_{\text{sp}}$ 个节点进行 MiniGo 训练任务，其中 N_{train} 是分布式训练新的模型的节点数量，N_{sp} 是自我博弈产生训练样本的节点数量。

2.3.1 分布式计算

分布式训练任务使用 N_{train} 个节点。节点间执行数据并行，如图 2 的①训练所示。首先将当前模型的样本文件夹中的样本打乱混洗、均匀分为 N_{train} 个片段。然后将样本传输到各个训练节点。各个节点读取相同的最新模型参数值后开始计算。经过前向和反向计算得到各参数的梯度。各节点的梯度值归约得到参数的更新值，再广播给各个训练节点，更新各自模型参数。像这样迭代指定步数。其中，为减少节点间梯度值归约计算时受通信带宽的影响，采用 Ring All-Reduce 方法更新参数。

分布式自我博弈任务使用 N_{sp} 个节点。节点间进行任务级并行，如图 2 的①自我博弈所示。每个节点使用最新模型，进行自我博弈，从而得到训练样本。产生的样本保存在最新模型对应的样本文件夹下，作为下次训练的训练数据集。

2.3.2 高效 I/O 部署

不同于传统高性能计算（high performance computing，HPC）应用，在多节点分布式 MiniGo 训练中，节点间会频繁进行数据交换。随着节点规模增大，这种瞬时的并发跨节点通信将成为性能瓶颈。

本文方法根据系统结构特性进行了高效 I/O 部署，将大规模节点间的瞬时通信转换成离散的即时通信。在自我博弈中，N_{sp} 个自我博弈节点完成 1 个对局即时写入共享存储中，将瞬时并发网络访问消峰，变成若干异步网络访问。在训练开始阶段，主训练节点从共享存储读取最近 5 个模型产生的所有对局文件，进行样本文件压缩再写回共享存储。N_{train} 个训练节点从共享存储中读取

对应样本文件，开始分布式训练。得到最新的模型后写入共享存储。自我博弈节点从共享存储读取最新模型进行自我博弈。如此循环，直到得到迭代足够多次的模型。通过更改 MiniGo 的文件读写行为完成瞬时并发网络消峰和冗余操作消除，实现了高效的 I/O 部署。

2.4　CPU-DSP 异构协同计算

CPU-DSP 异构协同计算可分为数据预取与流水并行、共享内存编码两个部分。

2.4.1　数据预取与流水并行

本节优化可分为数据预取操作和流水并行操作，如图 2 的②所示。

数据预取操作重叠了 CPU 数据准备和 DSP 算子计算时间。通过设置一个根据需求动态改变大小的缓冲池，多线程执行数据的入队到缓冲池中。训练开始时，模型直接从缓冲池提取数据。数据的加载和读取可以同时执行，没有阻塞，从而 I/O 的时间几乎可以忽略不计。

异构设备流水并行操作重叠了 CPU 数据处理和 DSP 算子计算时间。CPU 数据处理包括网络构建、节点间通信、归约计算、MCTS。以自我博弈任务为例，智能体下一步行动根据 MCTS 结果确定。MCTS 分为选择、扩展、推理、回溯，其中选择和推理需要训练模型的推理结果。因此，将复杂的计算任务分割，将网络算子计算卸载到 DSP 进行，其余在 CPU 进行，实现了异构设备间的流水并行。

2.4.2　共享内存编码

根据原型系统 CPU 和 DSP 有共享物理空间的特性设计了共享内存编程，同时提供了一个基于 TensorFlow 的共享内存编码模式下的算子异构计算库。

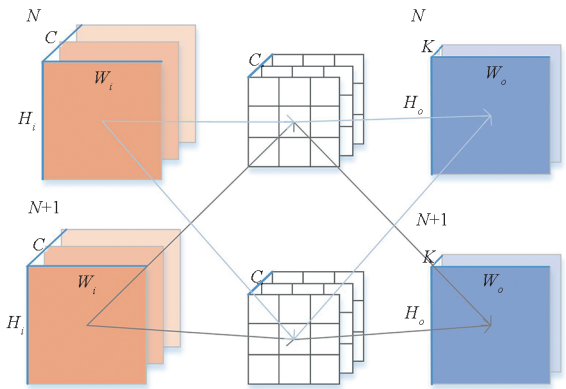
根据原型系统中存储管理硬件逻辑设计，修改 TensorFlow 等深度学习框架的算子内存管理，去掉 CPU-DSP 和 DSP-CPU 的 2 次数据传递操作，将数据存储在 CPU-DSP 共享存储段。变量、输入/输出等内存管理全周期都在共享物理地址空间，既无数据搬运的问题，也无须复杂的流水调度重叠通信和计算。同时，提供了易于使用的共享内存编码模式下的算子异构计算库。

2.5　DSP 并行

实现 DSP 核间并行以及 DSP 指令集并行可以加速算子计算。结合性能分析工具，发现 MiniGo 应用的计算热点集中在神经网络推理和训练过程，具体而言是卷积、批归一化（batch normalization，BN）等典型算子。为了降低 MiniGo 应用执行时间，针对上述算子进行了汇编优化，将其卸载到 DSP 中高效执行。

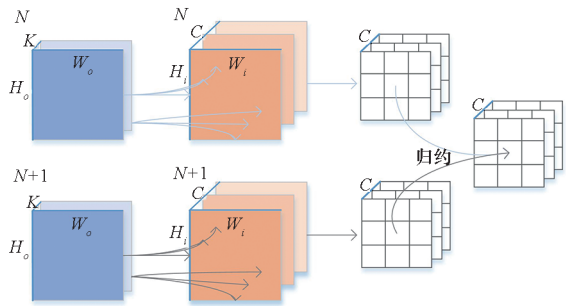
如图 2 中③所示，将输入数据均匀划分，实现 DSP 核间数据并行计算。如图 2 中③所示，充分开发了向量处理器的标量、向量协同数据加载能力，多计算资源并行计算能力，实现紧凑的流水线并行。

以 conv3 ×3 的直接卷积过程为例，其计算过程如图 3 所示。其中，图 3（a）为前向计算过程，图 3（b）为反向计算过程。在前向计算中，输入 feature 大小为 *NHWC*，卷积核大小为 3 ×3 × *C* × *K*，输出大小为 *NHWK*。其中，*N* 为批数据大小，*H* 为高，*W* 为宽，*C* 为通道数，*K* 为输出通道数。通过分析前向计算过程可知，其中 *N* 维度 batch 之间无计算相关性。可利用这个特点实现 DSP 簇内多核并行。对于每个 batch 为 *N*/24 的直接卷积，输出通道 *K* 维度之间无计算相关性，可实现 DSP 计算单元的硬件并行。在反向计算中，输入 feature 大小为 *NHWC*，输入 last_grad 大小为 *NHWK*，卷积核大小为 3 ×3 × *C* × *K*。输出是权重



（a）前向计算

（a）Forward calculation



（b）反向计算

（b）Backward calculation

图 3　conv3 ×3 的计算过程示意图

Fig. 3　Schematic diagram of calculation process of conv3 ×3

更新值和输入更新值，大小不变，分别是 $3\times3\times C\times K$ 和 $NHWC$。根据对其计算过程进行拆分，可以分为两个部分：第一，batch 之间无计算相关性的，利用 DSP 多核、核内多计算单元并行；第二，batch 之间存在的数据交互，则使用直接累加归约和二分累加归约。

前向计算在核上的映射过程为：将特征图从 DDR 加载到核内 SM 中，卷积核从 DDR 加载到核内 AM 中。核心计算为特征图的 1 个数据广播为 K 份，和卷积核 K 个数据的向量乘，得到 K 个累加结果。循环核心计算，累加 $3\times3\times C$ 次，得到 $1\times1\times K$ 个结果。然后再 H 和 W 上循环。设计实现的卷积并行方案如图 4。

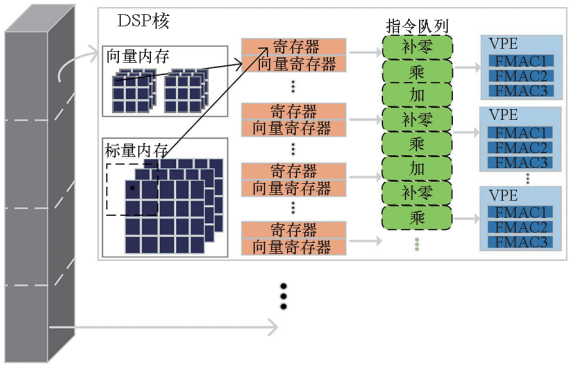


图 4 conv3 × 3 前向计算映射示意图

Fig. 4 Schematic diagram of conv3 × 3 forward calculation mapping

反向计算在核上的映射过程为：将特征图从 DDR 加载到核内 SM 中，卷积核和输入梯度值加载到核内 AM 中。核心计算是将一组输入梯度值放入向量寄存器（vector register，VR）中，特征图放入寄存器（register，R）中。将输入梯度值核特征图放入浮点乘累加操作器（float multiply accumulate，FMAC）中进行乘累加操作得到 $1\times1\times C\times K$ 的结果。如此平移循环 9 次，得到 1 组 $H_f\times W_f\times C\times K$ 的参数梯度矩阵。单个核内同时进行多组计算，得到多组参数梯度矩阵，将它们进行直接累加归约，然后放入 GSM。核间进行二分累加，得到最后的结果。

3 实验分析

为了测试本文方法的性能，做了以下实验设置。实验的数据集来自智能体自我博弈，每一次模型迭代对弈 8 192 局作为样本。训练停止条件为智能体与目标智能体对弈胜率超过 50%。如无特别说明，根据 3.1.1 实验分析选择 $N_{train}=43$，batchsize 取 96。其他超参数设置与 MLPerf 默认参数一致。

3.1 核心技术

3.1.1 节点间优化

节点间的优化主要表现在 I/O 优化和实现分布式训练与自我博弈。

1）I/O 优化：在大规模并行训练开始阶段时，大量节点同时进行数据操作会对共享文件系统产生很大的瞬时读写需求。在本小节展示了高效 I/O 部署前后训练开始阶段数据传输耗时对比。实验内容为记录在训练开始阶段，共享文件系统中数据读写时间。实验分为优化前和优化后两组，每组实验重复 3 次。I/O 性能对比如表 1 所示，重复实验得到的实验结果波动较大。这是由于在多租户场景下，受到了其他租户负载的影响。在优化前，最大耗时 11 195 s，最小耗时 2 730 s。在优化后，最大耗时 779 s，最小耗时 140 s。本文提出的高效 I/O 部署优化方法取得了显著效果。本文方法中的高效 I/O 部署优化有效支撑了 MiniGo 的大规模并行训练。

表 1 I/O 性能对比

Tab. 1 I/O performance comparison

类别	序号	开始	结束	时间间隔/s
优化前	Trace 1	11:33:13	14:39:48	11 195
	Trace 2	15:04:01	16:01:03	3 423
	Trace 3	10:13:14	10:58:44	2 730
优化后	Trace 4	16:58:49	17:10:50	721
	Trace 5	17:53:21	18:06:20	779
	Trace 6	09:14:03	09:16:23	140

2）分布式自我博弈：本小节讨论了 I/O 优化后，分布式自我博弈完成速度。表 2 展示了迭代 1 个模型，分布式自我博弈耗时随着节点数增加的变化。可以看到，1 024 个节点平均耗时 3 000 s。当节点数量翻 1 倍时，耗时为原来的一半，平均耗时约为 1 400 s。从实验结果可知，经

表 2 分布式自我博弈平均耗时分析

Tab. 2 Average time-consuming analysis of distributed self game

总样本数	节点数	平均时间/s
8 192	1 024	3 000
8 192	2 048	1 400
8 192	4 096	699

过 I/O 优化,节点数增加伴随的 I/O 操作增加没有影响自我博弈样本产生速度。自我博弈节点规模越大,自我博弈平均时间线性减少。

3)分布式训练:在本节讨论了分布式训练的时间(通信时间、计算时间、归约时间)与节点数的关系,如图 5 所示。固定总的 batchsize 都为 4 128,step 数为 241。当节点数减少,单节点 batchsize 增大,计算规模增大,计算时间随之增大。当节点数增大,通信量增大,多节点归约的通信时间随之增加。本文设置了 4 组实验。如图 5 所示,当训练节点数为 86、单节点 batchsize 为 48 时,与其他实验设置相比,总的训练时间最短,但通信时间最长。当训练节点数为 43、单节点 batchsize 为 96 时,网络开销小于其他参数设置实验。实验结果表明,节点数增加,step 的通信时间变化不大。通信时间不变是因为将算子计算卸载到 DSP 进行,使得 CPU 足够完成除算子计算以外的其他操作。所以,在后续实验中 batchsize 设置为 96,训练节点数量 $N_{\text{train}}=43$。

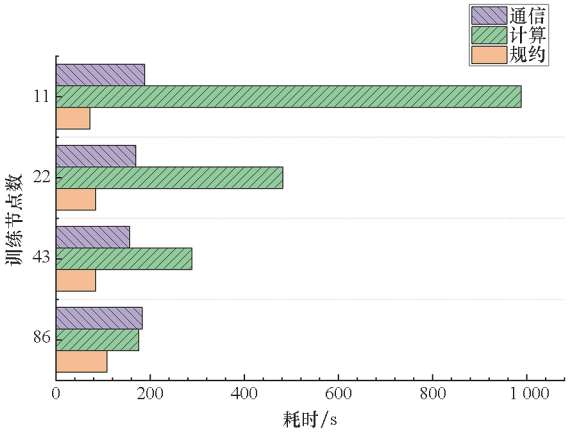


图 5　网络开销与节点数的关系

Fig. 5　Relationship between network overhead and number of nodes

3.1.2　CPU-DSP 异构计算优化

在本节中展示了 CPU-DSP 异构计算下的数据预取优化的性能提升,以及共享内存编码优化的性能提升。

1)数据预取:实验设置为使用单节点进行端到端的模型训练,顺序地加载和读取大小为 13 × 19 × 19 × 96 的 FP32 数据。数据预取优化性能对比如表 3 所示,CPU-DSP 执行异构流水并行重叠的时间为54. 55 ms。优化后加载和读取数据耗时从58. 9 ms降至 4. 36 ms,加速比高达 13. 5。从实验结果可知,数据预取的优化折叠了大部分数据加载和读取时间。

表 3　数据预取优化性能对比

Tab. 3　Performance comparison of data prefetching optimization

操作	优化前/ms	优化后/ms	折叠时间/ms	加速比
IteratorGetNext	58. 9	4. 36	54. 55	13. 5

2)共享内存编码:通过训练时间来衡量共享内存编程模型带来的性能提升。共享内存编码优化前后计算时间对比如表 4 所示,共享内存编程模型优化对算子计算速度提升明显。卷积算子分为四种,分别是:conv3 _ initial, conv3 _ residual, conv1_policy,conv1_value。conv3_initial 在反向计算优化前后加速比达 7. 22。conv _ residual 优化后,前向计算耗时从 39. 734 ms 降到 7. 257 ms,反向计算耗时从 66. 565 ms 降到 13. 284 ms。BN + ReLU 算子在网络中被反复调用了 13 次,前向计算和后向计算的优化前后加速比分别是 3. 75 和 3. 13。从整个训练过程来说,1 个步长,前向计算总耗时从1 619. 96 ms降至 398. 77 ms,后向计算总耗时从2 661. 06 ms降至 756. 57 ms。优化前后,计算时间加速比为 3. 71,考虑到归约和训练启动开销,训练时间的加速比为 2. 39。

表 4　共享内存编码优化前后计算时间对比

Tab. 4　Comparison of calculation time before and after shared memory coding optimization

计算类型	算子类型	算子数量	优化前/ms	优化后/ms	加速比
前向计算	conv3_initial	1	25. 293	7. 806	3. 24
	conv3_residual	12	39. 734	7. 257	5. 48
	conv1_policy	1	25. 024	2. 099	11. 92
	conv1_value	1	21. 848	2. 013	10. 85
	BN + ReLU	13	79. 899	21. 321	3. 75
反向计算	conv3_initial	1	28. 651	3. 966	7. 22
	conv3_residual	12	66. 565	13. 284	5. 01
	conv1_policy	1	69. 606	11. 195	6. 22
	conv1_value	1	58. 194	11. 045	5. 27
	BN + ReLU	13	127. 393	40. 636	3. 13

3.1.3　DSP 平台算子计算优化

在本节中,展示了 DSP 平台算子计算优化的性能提升。首先,分析了模型训练过程中各个算子所占用的训练时间。然后,将耗时占比大的算

子计算卸载到 DSP 进行,并展示了算子计算优化效果。

1)计算热点分析:使用单核 CPU 进行训练时,各个算子的计算时间占比如图 6 所示。各算子占比有较大差异。特别地,卷积计算消耗了训练中计算耗时的绝大部分,在前向计算中耗时占比是 90.88%,在反向计算中耗时占比是 90.66%。BN 计算时间占前向计算时间的 7.78%,占反向计算时间的 8.15%。ReLU 和全连接的前向、反向计算时间分别占比不到 1%。因此,本文优化了耗时占比大的算子计算。

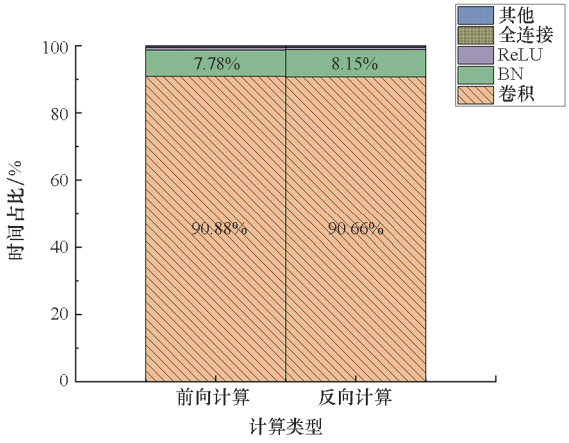


图 6 当只使用 CPU 进行训练时,各个算子在前向计算和反向计算中的耗时

Fig. 6 When only CPU is used for training, the time consumption of each operator in forward and backward calculation

2)算子优化:算子优化前后实验的设置为使用 1 个 DSP 进行优化后算子的计算,16 个 CPU 核进行未优化算子的计算。算子优化性能对比如表 5 所示,优化后的卷积算子计算速度都有不同程度的涨幅。对于 conv3_residual 的速度提升最为明显,在前向计算中加速比高达 8.3,在反向计算中加速比高达 16.44。BN + ReLU 算子性能不理想,原因在于 BN 和 ReLU 的计算十分简单,计算过程中,待计算矩阵保存在共享内存上,申请空间和通信的操作耗时大于 BN 和 ReLU 的计算耗时,导致负优化的情况。从整体看,1 个步长中,前向计算加速比为 1.67,反向计算加速比为 2.78,有明显的性能提升。在 MiniGo 模型训练中,算子计算发生在训练和自我博弈阶段。不同的是,在训练阶段前向计算和后向计算都要发生,在自我博弈阶段只发生前向计算进行推理。通过对算子进行优化,自我博弈中一次推理由 666.27 ms降到 398.77 ms,1 个步长计算总耗时由 2 766.98 ms 降到1 155.34 ms。

表 5 算子优化性能对比

Tab. 5 Comparison of operator optimization performance

计算任务	算子类型	优化前/ms	优化后/ms	加速比
前向计算	conv3_initial	16.24	7.81	2.08
	conv3_residual	60.27	7.26	8.30
	conv1_policy	2.68	2.10	1.28
	conv1_value	2.52	2.01	1.25
	BN + ReLU	16.06	21.32	0.75
反向计算	conv3_initial	18.63	3.97	4.70
	conv3_residual	218.36	13.28	16.44
	conv1_policy	32.85	11.20	2.93
	conv1_value	36.07	11.05	3.27
	BN + ReLU	26.52	40.64	0.65

3.2 总体优化效果

输入为训练模型、读模型样本窗口数、自我博弈阶段对弈局数、模型自我博弈产生样本、模型评估对局数以及目标模型,输出为与目标模型对弈超过 50% 胜率模型所需训练时间。本文完整地进行了连续训练,每次运行都超过10 000 步。在每一次运行中,网络初始化参数来自同一训练模型。重复进行了多次实验。训练的单节点 batchsize 固定为96,训练节点固定为43,自我博弈节点数从 1 024 扩展至 4 096。总体训练时间随节点数变化如表 6 所示,随着节点规模增大,迭代次数增大,训练时间减小。1 067节点训练时间约为 43.02 h,4 139 节点训练时间为 16.05 h。

表 6 总体训练时间随节点数变化

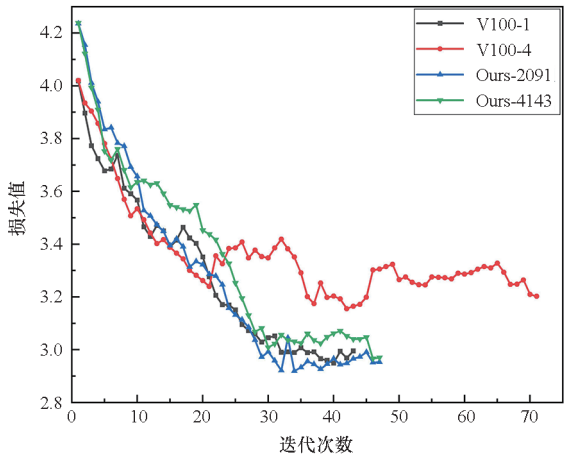
Tab. 6 Changes of overall training time with number of node

训练节点数量	自我博弈节点数量	迭代次数	时间/s
43	1 024	43	154 886
43	2 048	45	88 535
43	4 096	47	57 780

3.2.1 收敛性和稳定性分析

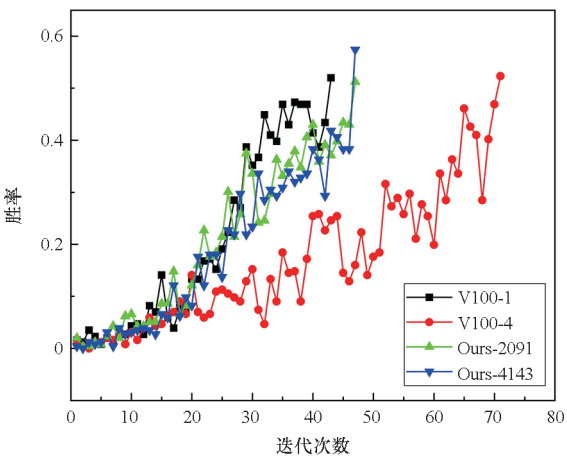
设置四组实验来对比多节点分布式训练的收

敛性。四组对比实验中总的 batchsize、自我博弈样本量等主要参数设置一致。V100-1 实验在单个 V100 上进行训练。V100-4 在 4 个 V100 上进行分布式训练。Ours-2091 为本文方法在2 091个节点上进行训练。Ours-4139 为本文方法在 4 139 个节点上进行训练。收敛表现和稳定性如图 7 所示。图 7(a)展示损失值随训练迭代次数变化。可以看到,节点数增多,损失值波动较大。整体趋势都是随着迭代次数增加,损失值减小。当迭代到 45 次以后,V100-4 持续波动,且损失值保持在 3.1 以上。相对应地,本文方法损失值逐渐稳定在 3 以下。图 7(b)展示胜率随训练迭代次数变化。V100-1 迭代训练 43 次达到目标,V100-4 需要迭代 74 次,Ours-2091 需要迭代 45 次,Ours-4139 需要迭代 47 次。



(a) 损失值随训练迭代次数变化

(a) Loss changes with the number of training iterations



(b) 胜率随训练迭代次数变化

(b) Winning rate changes with the number of training iterations

图 7　性能对比随训练迭代次数变化

Fig. 7　Performance comparison changes with the number of training iterations

随着节点数增大,达到目标模型所需迭代次数逐渐增大。原因在于,强化学习中样本效率很难保证。分布式自我博弈和分布式训练,使得学习变得更加困难。当节点规模达到 224 时,NVIDIA 方法的迭代次数为 65。V100-4 相较于 V100-1 滞后了 31 个模型。在本文方法中,节点规模达到 4 139 时,迭代次数为 47。Ours-4139 比 V100-1 仅仅滞后 4 次迭代,且节点规模翻倍对迭代次数几乎没有影响。本文方法在总体训练迭代次数上具有良好的稳定性。

3.2.2　可扩展性

参考强扩展性定义,将问题规模和迭代次数的关系进行归一化,即将达到 MLPerf 终止条件时的时间用迭代次数进行归一化,再计算随着节点规模增加时的可扩展效率。

选择固定训练节点数 $N_{train}=43$,通过调整自我博弈节点数量来缩减计算时间。表 7 展示了本文方法的可扩展效率对比。以 Ours-1067 作为基准,扩展系数为 1。当自我博弈节点规模增大 1 倍,总节点规模增大 1.959 倍时,训练所需时间是 88 535 s。与 Ours-1067 相比,Ours-2091 可扩展效率为 89.2%。与 Ours-1067 相比,Ours-4139 可扩展效率为 69.1%。

表 7　可扩展效率对比

Tab. 7　Comparison of the expanded efficiency

方法	节点规模	节点规模增率	时间/s	可扩展效率/%
Ours-1067	1 067	1.000	154 886	100.0
Ours-2091	2 091	1.959	88 535	89.2
Ours-4139	4 139	3.879	57 780	69.1
Ours-8235	8 235	7.717	43 120	46.5

在 MiniGo 应用中,自我博弈节点数 $N_{sp}=8\ 192$个能够完全展开任务级并行。因此,根据 N_{sp}取 1 000、2 000 和 4 000 的实验数据,训练和自我博弈的可扩展性实验数据,分析得到自我博弈平均时间和节点规模呈现近似线性关系,同时训练平均时间近似不变。通过稳定性分析,当节点规模变大后自我博弈时间减少,随着其和训练时间接近,由于训练样本中旧样本占比的提升,收敛速度会受到一定影响,训练迭代次数呈现等差上升趋势。当节点规模达到 8 235 时,自我博弈平均时间约为 350 s,迭代次数约为 49 次,训练所需时间约为 43 120 s,可扩展效率估计为 46.5%。

4 结论

本文提出了一种适用于大规模异构计算平台训练 MiniGo 智能体的高效多级并行训练方法。实现了 MiniGo 训练的充分并行,包括节点间任务级并行、CPU-DSP 异构并行、DSP 核内并行。根据 MiniGo 应用特点和原型系统特点设计了节点间任务级并行,同时也考虑到网络通信成为计算瓶颈的可能性,进行了高效的 I/O 部署。提出了面向 CPU-DSP 共享内存结构的异构计算内存管理,减少异构设备间的数据搬运。共享内存编程优化对算子计算速度提升明显,算子计算优化前后加速比最大是 11.92。同时,将网络中计算密集的算子卸载到 DSP 计算,保留充足的 CPU 资源完成其他任务。结合 DSP 硬件特点,充分利用片上存储带宽实现算子的高效计算。与 16 核 CPU 计算相比,单核 DSP 计算算子最大加速比达 16.44。从整个智能体训练过程来看,本文方法实现计算节点规模从 1 067 扩展至 4 139,得到达到给定终止条件所需时间从 43.02 h 降至16.05 h,可扩展效率为 69.1%。

参考文献(References)

[1] SILVER D, HUANG A, MADDISON C J, et al. Mastering the game of Go with deep neural networks and tree search[J]. Nature, 2016, 529(7587): 484 – 489.

[2] VINYALS O, BABUSCHKIN I, CZARNECKI W M, et al. Grandmaster level in StarCraft Ⅱ using multi-agent reinforcement learning[J]. Nature, 2019, 575(7782): 350 – 354.

[3] RUDIN N, HOELLER D, REIST P, et al. Learning to walk in minutes using massively parallel deep reinforcement learning[C]//Proceedings of the 5th Conference on Robot Learning, 2022: 91 – 100.

[4] ZHANG R, LV Q B, LI J, et al. A reinforcement learning method for human-robot collaboration in assembly tasks[J]. Robotics and Computer-Integrated Manufacturing, 2022, 73: 102227.

[5] LI J W, YU T, ZHANG X S. Coordinated load frequency control of multi-area integrated energy system using multi-agent deep reinforcement learning[J]. Applied Energy, 2022, 306: 117900.

[6] WURMAN P R, BARRETT S, KAWAMOTO K, et al. Outracing champion Gran Turismo drivers with deep reinforcement learning[J]. Nature, 2022, 602(7896): 223 – 228.

[7] DENG Z H, FU Z Y, WANG L X, et al. False correlation reduction for offline reinforcement learning[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2024, 46(2): 1199 – 1211.

[8] ZHONG Y F, Kuba J G, FENG X D, et al. Heterogeneous-agent reinforcement learning[J]. Journal of Machine Learning Research, 2024, 25(32): 1 – 67.

[9] MNIH V, KAVUKCUOGLU K, SILVER D, et al. Human-level control through deep reinforcement learning[J], Nature, 2015, 518: 529 – 533.

[10] BERNER C, BROCKMAN G, CHAN B, et al. Dota 2 with large scale deep reinforcement learning[EB/OL]. (2019 – 10 – 13) [2022 – 01 – 20]. https://cdn. openai. com/dota – 2. pdf.

[11] MATTSON P, REDDI V J, CHENG C, et al. MLPerf: an industry standard benchmark suite for machine learning performance[J]. IEEE Micro, 2020, 40(2): 8 – 16.

[12] ONG H Y, CHAVEZ K, HONG A. Distributed deep Q-learning[EB/OL]. (2015 – 10 – 15)[2022 – 01 – 20]. http://arxiv. org/abs/1508.04186v2.

[13] DEAN J, CORRADO G S, MONGA R, et al. Large scale distributed deep networks[C]//Proceedings of the 25th International Conference on Neural Information Processing Systems, 2012.

[14] NAIR A, SRINIVASAN P, BLACKWELL S, et al. Massively parallel methods for deep reinforcement learning[EB/OL]. (2015 – 07 – 16)[2022 – 01 – 21]. http://arxiv. org/abs/1507.04296v2.

[15] MNIH V, BADIA A P, MIRZA M, et al. Asynchronous methods for deep reinforcement learning[EB/OL]. (2016 – 06 – 16) [2022 – 01 – 21]. http://arxiv. org/abs/1602.01783v2.

[16] HORGAN D, QUAN J, BUDDEN D, et al. Distributed prioritized experience replay[EB/OL]. (2018 – 03 – 20)[2022 – 01 – 21]. http://arxiv. org/abs/1803.00933v1.

[17] ESPEHOLT L, SOYER H, MUNOS R, et al. IMPALA: scalable distributed deep-RL with importance weighted actor-learner architectures[C]//Proceedings of the 35th International Conference on Machine Learning, 2018.

[18] LIANG E, LIAW R, MORITZ P, et al. RLlib: abstractions for distributed reinforcement learning[C]//Proceedings of the 35th International Conference on Machine Learning, 2018.

[19] MORITZ P, NISHIHARA R, WANG S, et al. Ray: a distributed framework for emerging AI applications[EB/OL]. (2018 – 09 – 30)[2022 – 01 – 21]. https://arxiv. org/abs/1712.05889.

[20] SERGEEV A, DEL BALSO M. Horovod: fast and easy distributed deep learning in TensorFlow[EB/OL]. (2018 – 02 – 21) [2022 – 03 – 02]. http://arxiv. org/abs/1802.05799v3.