doi:10.11887/j.cn.202401014 http://journal.nudt.edu.cn

高性能异构加速器 MiniGo 算子优化方法

乔 鹏[1,2],贺周雨[1,2],李荣春[1,2*],姜晶菲[1,2]
(1. 国防科技大学 计算机学院,湖南 长沙 410073;
2. 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:根据高性能异构加速器的特性和 MiniGo 的训练模式提出了一种高效的并行计算方法。对片上计算资源进行合理规划,实现异构设备之间的流水并行优化;根据异构设备间存在共享存储段设计了共享内存编码模式,减少数据传输开销;根据数字信号处理簇内具有多计算资源的特点结合算子计算-访存特性设计了不同的算子并行计算优化策略。同时,面向 TensorFlow 实现了一个易于使用的高性能计算库。实验结果显示,该方法实现了典型算子的多核并行计算。相对于单核,卷积算子加速比为 24.69。相较于裁剪版 8 核 FT2000+CPU,该方法训练和自博弈执行速度加速比分别为 3.83 和 1.5。

关键词:异构计算;算子优化;卷积神经网络;强化学习
中图分类号:TP391 **文献标志码**:A **开放科学(资源服务)标识码(OSID)**:
文章编号:1001-2486(2024)01-131-10

听语音 与作者互动 聊科研

Optimizing operator computation of MiniGo on high-performance heterogeneous accelerator

QIAO Peng[1,2], *HE Zhouyu*[1,2], *LI Rongchun*[1,2*], *JIANG Jingfei*[1,2]
(1. College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China;
2. National Key Laboratory of Parallel and Distributed Computing, National University of Defense Technology, Changsha 410073, China)

Abstract: An efficient parallel computing method based on the characteristics of the high-performance heterogeneous accelerator and the training mode of MiniGo was proposed. The on-chip computing resources were reasonably planned to achieve pipelining parallel optimization between heterogeneous devices. The shared memory programming was designed according to the existence of shared storage segments between heterogeneous devices to reduce data transmission costs. According to the characteristics of multiple computing resources in a digital signal processing cluster, combined with the computing-memory access feature of the operators, different optimization strategies were designed. At the same time, this method provides an easy-use high-performance operator library for TensorFlow. The experimental results show that this method realizes the multi-core parallel computing of operators. The speedup of convolution was 24.69 compared with that was achieved on a single core. Compared with the cropped version of the 8-core FT2000+ CPU, the speedup of training and self-play execution on this method were 3.83 and 1.5, respectively.

Keywords: heterogeneous computing; operator optimization; convolutional neural networks; reinforcement learning

目前,卷积神经网络(convolutional neural networks,CNNs)在人工智能典型应用领域取得了令人印象深刻的效果[1-6]。结合深度学习和强化学习,深度强化学习(deep reinforcement learning,DRL)被认为是处理决策类问题的有效方法之一。DRL 网络模型的推理和训练需要大量的计算和外部存储访问,再加上智能体和仿真环境交互中存在复杂的计算过程,这导致得到一个合格的智能体需要巨大的算力。计算模式复杂、对算力需求高的特点,使得 DRL 算法适合作为大规模计算平台的系统性能评测应用。因此,MLPerf[7] 机器学习基准测试应用集中包含了强化学习分区。MLPerf 是一套测量机器学习训练和推理在特定软硬件上性能表现的基准测试应用集,其中强化学习分区基准测试应用是 MiniGo。MiniGo 训练过程中,需要收集大量自我博弈的样本,因此需要面向大规模计算集群进行部署和优化。通过分析提交结果和代码,在单结点对 MiniGo 中策略网络

收稿日期:2022-12-15
基金项目:国家重点实验室稳定支持资助项目(WDZC20205500104)
第一作者:乔鹏(1988—),男,河北保定人,助理研究员,博士,E-mail:pengqiao@nudt.edu.cn
***通信作者**: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

进行推理和训练优化是十分重要的。因此,本文研究了 MiniGo 与自研高性能异构加速器的定制化适配。①结合加速器系统特性对 MiniGo 算子进行定制化汇编优化,在多核、多向量处理单元(vector processing element, VPE)、多寄存器、多指令间实现并行计算。②在中央处理器(central processing unit, CPU)和数字信号处理器(digital signal processing, DSP)的异构计算层次,提出一种设备间共享内存编码模式,有效减少片内数据搬运开销。同时,在设备间实现流水并行。③提供一个易使用的面向 TensorFlow 的高性能算子计算库。高性能算子计算库包含了 DSP 驱动和 CPU-DSP 异构执行引擎。

1 相关工作

1.1 MiniGo

强化学习普遍对算力需求高,适合应用于高性能异构加速器定制化适配的测试。2013 年 DeepMind 提出深度 Q-Learning 网络[1],融合深度学习与传统强化学习 Q-Learning 方法,在 Atari 中取得了突破性进步。随后 AlphaGo[2]、AlphaStar[3]、OpenAI Five[4]、AlphaFold[5] 等在围棋、即时战略游戏等领域取得里程碑式的突破。围棋在很长一段时间被认为是人工智能领域最具挑战的经典游戏之一。围棋第一手有 361 种下法,第二手有 360 种,第三手有 359 种,依次类推,一共有 361! 种下法,考虑到存在大量不合规则的棋子分布,合理的棋局约占 1.2%,约为 2.08×10^{170}[8]。由于搜索空间巨大,围棋存在难以落子的问题。AlphaGo 开创性地结合神经网络和蒙特卡罗树搜索(Monte Carlo tree search, MCTS),通过近似估值函数估计对弈结果降低了搜索深度,利用基于策略函数的采样动作降低了搜索广度。经过有监督学习训练、自我博弈强化学习,以 99.8% 胜率打败所有围棋程序,以 5 局完胜的成绩打败围棋冠军选手。MLPerf 选择 MiniGo 作为基准测试应用。MiniGo 是根据 AlphaGo Zero[9] 编写的开源围棋智能体训练代码。

MiniGo 智能体训练主要分为训练和自我博弈。训练任务中,使用最近的训练得到的模型和其自我博弈产生的样本进行训练,得到新的模型。自我博弈任务中,使用最新模型进行自我博弈,产生新的样本,作为下一次训练的训练样本。两个任务互相依赖,交替进行。训练阶段主要包括数据传递、网络层调度、网络的前向推理和反向更新。自我博弈阶段通过 MCTS 计算落子位置。MCTS 分为选择、扩展、评估、回溯四个部分,其中选择、扩展、评估需要训练模型的推理结果。在自博弈的评估阶段会进行前瞻,通过假想对局来估计当前动作的价值。以 1 000 个对局为例,平均每 1 个对局执行前瞻 101 884 次,1 个对局内最大前瞻次数为 225 318。其中每次前瞻都会执行 1 次模型的推理。在 1 个 8 核的 i7-9700K 训练智能体。实验结果显示,在训练阶段,网络的前向推理和反向更新占总时间的 71.3%。在自博弈阶段,网络的前向推理占总时间的 83.26%。由此可见,模型训练中计算热点在于神经网络算子的计算。它的算子构成如表 1 所示。

表 1 MiniGo 网络结构

Tab. 1 Network structure of MiniGo

算子	输入	权重	输出
CONV3_INIT	(19,19,13)	(3,3,13,64)	(19,19,64)
CONV3_RES	(19,19,64)	(3,3,64,64)	(19,19,64)
CONV1_POLICY	(19,19,64)	(1,1,64,2)	(19,19,2)
CONV1_VALUE	(19,19,64)	(1,1,64,1)	(19,19,1)
FC1	(1,722)	(722,362)	(1,362)
FC2	(1,361)	(361,64)	(1,64)
FC3	(1,64)	(64,1)	(1,1)

目前,向 MLPerf 榜单提交 MiniGo 训练结果的机构有 NVIDIA 和 INTEL。NVIDIA 设计了一个自研 CPU-GPU 异构计算系统(DGXA100-NGC20.06)训练 MiniGo。NVIDIA 提供的方法在单结点上的优化主要是卷积神经网络的低精度计算。INTEL 设计了一个自研的纯 CPU 计算系统(4socket-per-node-CPX-6UPI)。INTEL 提供的方法在训练时采用单精度浮点计算以保证训练收敛,推理时采用 INT8 精度计算以保证推理速度。

加速器上的计算资源和存储容量是有限的,优化算子计算映射过程能够有效地提高资源利用率[10]。此外,芯片内外通信的高成本是实现更高性能的另一个主要障碍。大量数据移动和内存访问相关的能量消耗可能会超过计算的能量消耗[11-12]。

结合以上分析,本文方法在异构设备间计算任务分配、异构设备间数据通信、加速器算子计算三个方面进行了优化。

1.2　加速芯片

面对日益增加的算力需求,将计算密集和数据密集的网络计算卸载到加速器处理是一个有效的解决方法。常用的加速器有图形处理器(graphics processing unit, GPU)、张量处理器(tensor processing unit, TPU)、现场可编程门阵列(field programmable gate array, FPGA)、DSP 等。它们各自拥有不同的优势。例如,FPGA 相对于传统专用集成电路(application specific integrated circuit, ASIC)具有更强的可操作性,在相同功耗效率下可以达到比多核 GPU 更低延迟的推理。GPU 具有可用于较好的软件优化生态环境以及高功耗下的高性能优势。TPU 是专为张量(Tensor)计算而设计的加速器。相对于 FPGA 的低计算能力和 GPU 的高功耗,基于 DSP 的矢量处理器取得了性能和功耗的平衡。近年来,加速器的并行计算能力、内存容量、内存速度等都得到了极大的增强。除了硬件设备的改进,相应的面向加速器的算子计算方法也在改进[13-17]。

面对日益增加的算力需求,结合人工智能应用和计算平台特点进行高效适配是一个有效的解决方法[18]。实验显示,计算设备对于不同人工智能应用展现出的利用率不同。一些实验[19-20]表明,面向特定硬件的特定算法的协同优化设计能表现出更好的性能。

目前,面向加速器的算子加速方法研究得到了瞩目的发展。但依然存在一些挑战。首先是针对卷积神经网络的核心计算单元(如通用矩阵乘)在计算过程中如何合理使用特定加速器的多计算资源;然后是如何减少数据在加速器中传输占据的计算时间;最后是如何利用异构设备进行设备间并行计算。本文提出了面向高性能异构加速器的算子优化方法以解决上述问题。

1.3　自研原型系统

本文方法所依托的计算系统为 FT-M7032,如图 1 所示。该系统是国防科技大学面向 E 级计算自主研发的一款异构计算系统,由大容量片外存储、CPU、DSP 簇、内存双倍数据速率(double data rate, DDR)和其他外设组成。计算设备包含 1 个 16 核 ARMv8 CPU 和 4 个 DSP 簇。16 核 CPU 是裁剪版的 Phytium FT-2000 + Processor[21-22]。CPU 和 DSP 簇之间共享内存空间。大容量的片外存储和片内内存通过高速网络连接。

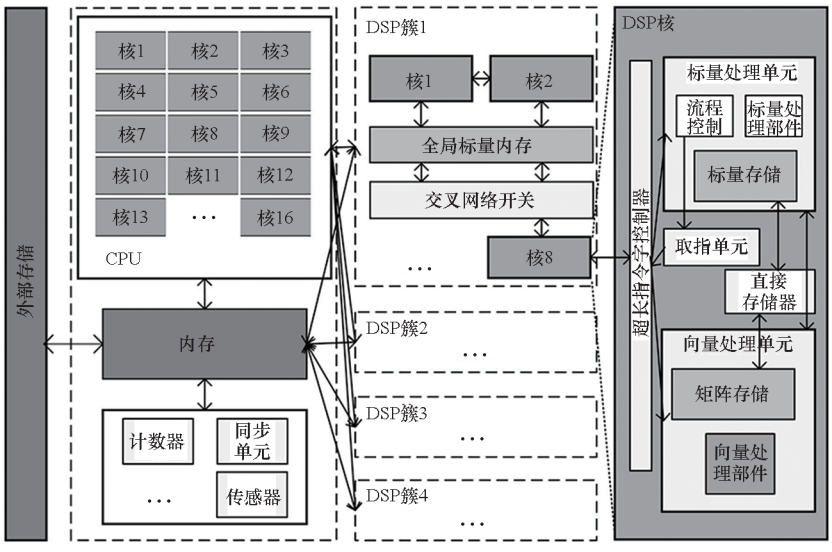


图 1　原型系统架构图

Fig. 1　System architecture diagram of prototype system

FT-M7032 拥有 4 个 DSP 簇,每个簇包含 8 个 DSP 核,1 个 6 MB 的片上全局共享存储(global shared memory, GSM)。GSM 片上带宽大约为 307.2 GB/s,内部包含 4 个子通路,可作为数据或指令的片上内存使用。单个 DSP 核在1.8 GHz工作主频下可以提供 345.6 GFLOPS 的峰值性能。

DSP 核主要由标量处理单元(scalar processing unit,SPU)、向量处理单元(vector processing unit, VPU)、直接存储器(direct memory access,DMA)、取指单元和超长指令字控制器组成。VPU 负责向量计算,主要由 16 个向量处理部件(vector processing element, VPE)与 768 KB 向量存储(array memory, AM)构成。每个 VPE 有 64 个 64 bit的寄存器和 3 个浮点乘累加(float multiply accumulate, FMAC)单元。16 个 VPE 一次可以处理 32 个单精度浮点数据。AM 与向量寄存器之间

的带宽为 921.6 GB/s。标量存储(scalar memory, SM)与寄存器之间的访问带宽为 28.8 GB/s。AM、SM 的数据可以通过 DMA 搬出或搬入。

2 MiniGo 算子加速方法

传统算子加速方法通常具有以下问题:①未对不同尺寸算子进行定制化优化,导致资源的浪费;②数据在设备间传递消耗大;③没有考虑异构设备的特性。针对上述问题,提出了一种高效的并行计算策略。在 DSP 端进行算子计算优化,为 MiniGo 的算子进行定制化汇编实现。结合原型系统具有异构计算设备、设备间具有共享内存段、MiniGo 算法流程的特性实现了异构设备间的高效并行计算。提供了面向 TensorFlow 的一种类英伟达统一计算架构(compute unified device architecture, CUDA)的高性能算子库,高效调用上述优化的计算。

2.1 DSP 端算子计算

FT-M7032 的 1 个 DSP 簇包含 8 个 DSP 核,可以通过 DSP 核间并行、核内并行、DSP 指令集并行加速算子计算。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,即全连接、卷积算子、批量规一化(batch normalization, BN)和修正线性单元(rectified linear unit, ReLU)等算子。为了降低 MiniGo 应用执行时间,针对这些算子进行了汇编优化,卸载到 DSP 中高效执行。

本节中数据格式采用 TensorFlow 默认格式 NHWC 和 NHWK 进行说明。其中,$\boldsymbol{I}$ 表示输入张量,$\boldsymbol{W}$ 表示权重张量,$\boldsymbol{O}$ 表示输出张量,N 是批大小,H 是张量高,W 是张量宽,C 是输入张量的通道数,K 是输出张量的通道数。

2.1.1 全连接算子

针对 MiniGo 的算子大小、自研系统的特性进行了定制化设计。

以 MiniGo 中的 FC2 为例。原始全连接算子的伪代码如算法 1 所示,本文全连接算子的伪代码如算法 2 所示。如算法 1 所示,在原始全连接算子中,内存访问操作数为 4NCK 次。其中,NCK 为循环次数,4 为矩阵 $\boldsymbol{O}$ 读取内存操作、计算后存储到内存操作以及 $\boldsymbol{IW}$ 的内存读取操作。完成一次全连接算子计算操作为 NCK 次乘累加操作。本文方法通过结合自研系统的软硬件特性,设计出针对性的优化方法。分析可并行性:对于前向计算,N 维度批次之间无计算相关性;输入的一行和权重的一列做乘累加时,在 K 维上无计算相关性;输入的一行可以进一步划分为多个块,并行地与对应权重相乘,保留中间结果,后续再累加。原型系统拥有多个 DSP 核,可以实现 N 维的并行。原型系统拥有多个 VPE 和寄存器,可以实现 K 维的并行、块划分的并行和指令向量化寄存器的并行。如算法 2 所示,在本文方法中,全连接算子的内存访问操作数为 4,分别是 $\boldsymbol{IWO}$ 的内存读取和 $\boldsymbol{O}$ 的内存写入操作。最内循环计算是(1,19)和(19,64)的矩阵乘法,使用向量化和寄存器化后将原本的 $1\times19\times19\times64$ 个乘累加指令缩减为 19 个。经过二次维度拆分并行化、矩阵乘向量化后,原始全连接算子串行的 NCK 次操作转换为并行的$N(C/19)$个 19 次操作。

算法 1 原始全连接算子前向计算伪代码

Alg. 1 Pseudo code for original fully connected operator in forward calculation

输入:输入矩阵 I[N][C],权重矩阵 W[C][K]
输出:输出矩阵 O[N][K]

```
1. Set N = 8, C = 361, K = 64, c = 19
2. for i = 0 : N do
3.     for k = 0 : K do
4.         O[i][k] = 0
5.         for j = 0 : C do
6.             O[i][k] += I[i][j] * W[j][k]
```

算法 2 本文方法全连接算子前向计算伪代码

Alg. 2 Pseudo code for this method's fully connected operator in forward calculation

输入:输入矩阵 I[N][C],权重矩阵 W[C][K]
输出:输出矩阵 O[N][K]

```
1. Set N = 8, C = 361, K = 64, c = 19
2. for i = 0 : N do (in parallel)
3.     for j = 0 : c : C do (in parallel)
4.         DMA(I[i][c] -> I_s[i][c]), DDR -> SM
5.         DMA(W[c][K] -> W_a[c][K]), DDR -> AM
6.         O_{vr}[i][K] = I_s[i][c] * W_a[c][K]
7.         Accumulate, O_{vr}
8. DMA(O_{vr}[N][K] -> O_{a}[N][K]), VR -> AM
9. DMA(O_{a}[N][K] -> O[N][K]), AM -> DDR
```

本文方法在片上的具体映射如图 2 所示,流程为:将 $\boldsymbol{I}[N][C]$ 在 N 维进行划分,划分后的矩阵以标量的方式从 DDR 加载到 DSP 核的 SM 中。对 $\boldsymbol{W}[C][K]$ 进行广播,以向量的形式从 DDR 加载到 DSP 核内的 AM 中。对 SM 中的数据根据块尺寸进行划分,划分后一组数据大小为(1,19)存

入寄存器 R 中,后续广播到向量寄存器(vecter register, VR)中。将一组数据与权重进行乘累加,得到一组结果放入 VR 中等待累加,如此循环 19 次完成 N 维上的 1 组数据,放入 AM 中,后续再通过 DMA 传回 DDR。如此在 N 维循环 8 次,得到全连接算子前向计算的结果 $\boldsymbol{O}[N][K]$。反向过程类似,在 N 维并行地在多核计算。每个核中处理一组数据,循环 19 次乘累加。在最后一次累加后将结果传入 GSM 中。由此得到 8 组数据,对位相加得到权重的更新值。

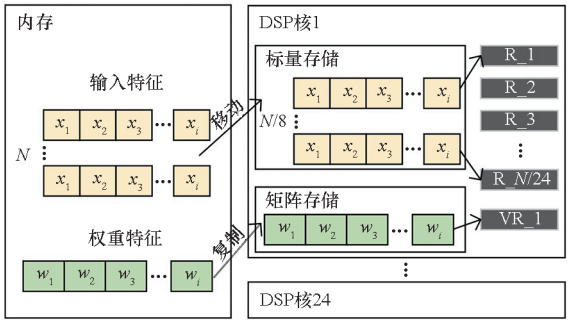


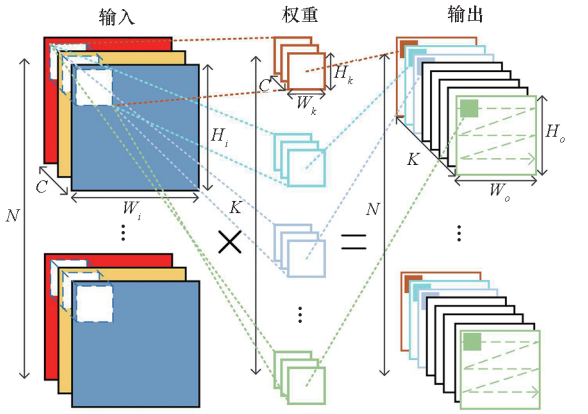
图 2　全连接计算片上映射示意图

Fig. 2　Schematic diagram of on-chip mapping for fully connected computation

2.1.2　卷积算子

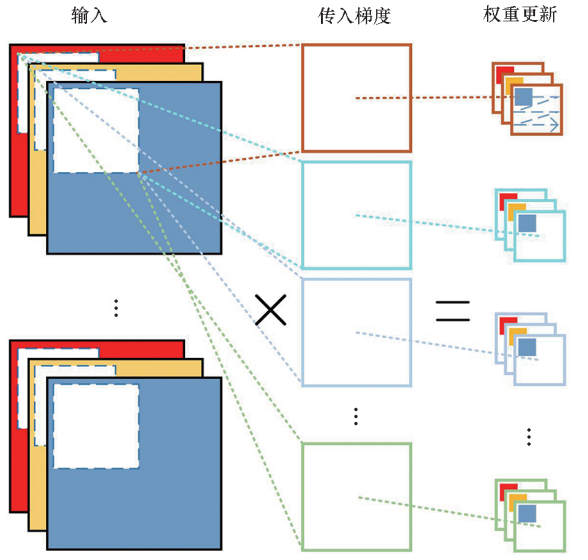
卷积是 MiniGo 中最耗资源的运算,将卷积计算卸载到 DSP 进行,能够有效加速计算。常用的卷积计算并行方案有 im2col、傅里叶卷积、winograd、直接卷积。im2col 会导致输入矩阵膨胀,恶化访存。傅里叶卷积增大了输入和卷积核大小,从而增加了内存带宽的需求。傅里叶卷积计算更复杂,涉及复数乘法,不适用于小卷积核。winograd 中转换映射的矩阵越大,计算精度的损失越大。结合 DSP 结构设计和 MiniGo 网络设计,本文方法选择了直接卷积的实现方法。

直接卷积涉及输入特征图和核权值的三维乘积累加运算,计算过程如图 3 所示。其中,图 3(a)为前向计算过程,图 3(b)为反向计算权重值过程。根据卷积计算的特性,分析可并行性。通过分析前向计算过程可知,N 维度批次之间无计算相关性。可利用这个特点实现 DSP 内多核并行。对于每个批次为 $N/8$ 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。对反向计算过程进行分析。根据对其计算过程进行拆分,可以分为 2 个部分:①批次之间无计算相关性的,利用 DSP 多核、核内多计算单元并行;②批次之间存在数据交互的,则使用直接累加归约和二分累加归约。



(a) 前向计算过程

(a) Forward computation process



(b) 反向计算过程

(b) Backward computation process

图 3　直接卷积计算过程

Fig. 3　Computation process of direct convolution

MiniGo 中的卷积算子根据数据规模可分为 CONV3_INIT, CONV3_RES, CONV1_POLICY, CONV1_VALUE。以 CONV3_INIT 为例,卷积前向计算在原型系统中的伪代码如算法 3 所示。在 N 维上划分 $\boldsymbol{I}$,将数据均匀分到 8 个核上并行计算。复制 8 份 $\boldsymbol{W}$ 广播到各个核内的 AM 中。在 K 维和 C 维上做循环,将一组大小为(3,3)的 $\boldsymbol{W}$ 放入 VR 中等待计算。在 $\boldsymbol{H}$ 和 $\boldsymbol{W}$ 维做循环,确定 $\boldsymbol{I}$ 的中心点,将中心点坐标放入 R1 和 R2 中。判断当前中心点是不是边界,如果是,则进行 padding。取中心点周围 9 个数据,存入 VR 中。

在 VR 中进行矩阵乘,得到 $\boldsymbol{O}$ 的 1 个值。将输出从 VR 存入 GSM。如此进行循环,得到 $\boldsymbol{O}$。

算法3 CONV3_INIT 前向计算伪代码

Alg. 3 Pseudo code for CONV3_INIT operator in forward calculation

输入:输入矩阵 I[N][H][W][C],权重矩阵 W[H][W][C][K]

输出:输出矩阵 O[N][H][W][K]

```
1. Set N = 8, H = 19, W = 19, C = 13, K = 64
2. for n = 0 : N do (in parallel)
3.    广播DMA(W[H][W][C][K] -> W_a[H][W][C][K])
4.    DMA(I[n][H][W][C] -> I_s[n][H][W][C]), DDR -> SM
5.    for k = 0 : K do  (in parallel)
6.       for c = 0 : C do
7.          DMA(W_a[3][3][c][k] -> W_{vr}[3][3][c][k])
8.          for h = 0 : H do
9.             for w = 0 : W do
10.               save h to R1, save w to R2
11.               if R1 or R2 edge (in parallel):补零
12.               DMA(I_s[n][3][3][c] -> I_{vr}[n][3][3][c])
13.               O_{vr}[n][h][w][k] = I_{vr}[n][3][3][c] x W_{vr}[3][3][c][k]
14.               DMA(O_{vr}[n][h][w][k] -> O_g[n][h][w][k])
15. DMA(O_g[N][H][W][K] -> O[N][H][W][K]), GSM -> DDR
```

把 $\boldsymbol{O}$ 通过 DMA 输出到 DDR 中。分析上述计算过程的并行性。在算法3的步骤2利用8个核在 N 维上实现并行。在步骤5利用多 VPE 和多寄存器,在 K 维实现并行。在步骤11,乘累加需要6个指令周期,跳转指令需要7个指令周期,同时进行这2个指令,从而覆盖部分进行边界判断的消耗。本文方法的卷积算子在汇编过程中利用多核、多 VPE、多寄存器、指令周期重叠实现了并行计算。

2.1.3 BN + ReLU

BN 有助于训练更快地收敛并防止网络过拟合。对于1个批次的输入 $\boldsymbol{X}=[x^{(1)},x^{(2)},\cdots,x^{(m)}]$,$\mu$ 是 $\boldsymbol{X}$ 的均值,δ^2 是 $\boldsymbol{X}$ 的方差。输出 $\boldsymbol{Y}=[y^{(1)},y^{(2)},\cdots,y^{(m)}]$。BN 前向计算公式为:

$$y^{(i)}=\gamma\frac{x^{(i)}-\mu}{\sqrt{\delta^2+\varepsilon}}+\beta \tag{1}$$

式中,γ 和 β 分别是通道尺度和偏差。在 MiniGo 中激活函数是 ReLU,如式(2)所示。

$$g(x)=\begin{cases}x, x>0\\ ax, x\leqslant 0\end{cases} \tag{2}$$

其中,a 等于0。当 x 大于0时,不变;当 x 小于等于0时,x 置0。

本文方法将 BN 和 ReLU 合并为1个算子,能够有效减少数据在片上的频繁读写。BN + ReLU 计算的输入数据大小为 $NHWC$。在 N 维可以使用多核并行计算。N 维之间存在数据交互,使用直接累加归约和二分累加归约。单个 DSP 核内,在 C 维进行 VR 级并行。C 是64,是 VR 处理数据能力的倍数。将单核内拆分后的数据组累加后保存在 AM 中,再把每个核的累加结果存储到 GSM 中。在 BN 计算后进行 ReLU 计算。对于正向计算,直接对 BN 的输出进行判定,当前值小于0则赋值为0,反之不变。反向计算时,如果当前位置的正向 ReLU 的结果为0,则将该位置上对应的传入梯度赋值为0,反之不变。

2.2 异构设备间并行

原型系统由1个16核 CPU、4个 DSP 簇、1个异构设备共享的 DDR 构成 CPU-DSP 异构计算节点,如图1所示。因此,可以依托 CPU-DSP 异构计算整合计算资源完成一个任务,以适配 MiniGo 的复杂计算需求。

2.2.1 流水并行

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、规约计算、蒙特卡罗树搜索。以自我博弈任务为例,智能体下一步行动根据蒙特卡罗树搜索结果确定。蒙特卡罗树搜索分为选择、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,本文方法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。流程为:DSP 完成算子计算后将结果返回给 CPU 进行蒙特卡罗树搜索。CPU 进行蒙特卡罗树搜索的同时,DSP 开始下一批次的算子计算。

2.2.2 共享内存编码

在 CPU-GPU 异构计算系统中,CPU 侧内存数据首先通过 CPU 调度经过总线和接口标准(peripheral component interface express, PCIE)总线传输到 GPU 侧显存中。GPU 计算后,将结果从显存再经 PCIE 总线传输到内存,完成一次计算卸载。为了减少数据搬运,TensorFlow 等框架一般会把所有的算子参数都放到显存,只在数据预取有一次 CPU-GPU 的搬运。但对于增量式搬运,则需要更多的 CPU-GPU 数据搬运,或者通过重叠

减少数据搬运的影响。原型系统中,DDR 中存在 CPU-DSP 共享的物理地址空间。因此,根据原型系统这一特性设计了共享内存编程。张量中权值变量、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,也无复杂的流水调度重叠通信和计算。根据原型系统的共享内存体系结构设计了共享内存编码模式。对于整个神经网络的前向、反向计算过程,可以减少数据在 CPU 和 DSP 之间的拷贝,从而减少了计算时间。

2.2.3　高性能算子库

为了让 TensorFlow 等深度学习框架使用 DSP 实现高性能的神经网络推理和训练,设计了类英伟达 CUDA 的高性能算子库,如图 4 所示。高性能算子库包含了 DSP 驱动和 CPU-DSP 异构执行引擎。高性能算子库将优化后的底层算子实现封装,给 TensorFlow 等深度学习框架提供统一的接口,供上层应用灵活使用。如图 4 所示,左边是调用 TensorFlow 库实现网络构建,右边是调用本文方法算子库实现网络构建。

```
import tensorflow as tf

...
def network():
    conv1 = functools.partial(
        tf.layers.conv2d,
        filter=...,
        kernel_size=...,
        padding=...,
        use_bias=...,
        data_format=...,
        name=...)

    layer1 = conv1(
        features=input_feature,
        filters=...,
        name=...)
...
```

```
import tensorflow as tf
from minigo_ops import convolution_mg
...
def network():
    conv1 = functools.partial(
        convolution_mg,
        filter=...,
        kernel_size=...,
        padding=...,
        use_bias=...,
        data_format=...,
        name=...)

    layer1 = conv1(
        features=input_feature,
        filters=...,
        name=...)
...
```

图 4　算子库调用示例

Fig. 4　Example of calling the operator library

3　实验

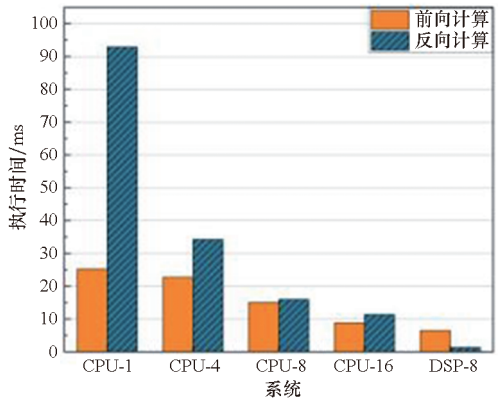
本节通过实验测试本文方法的性能。如无特别说明,输入数据的大小均为(96,19,19,13),数据类型为单精度浮点数。

3.1　DSP 端算子优化

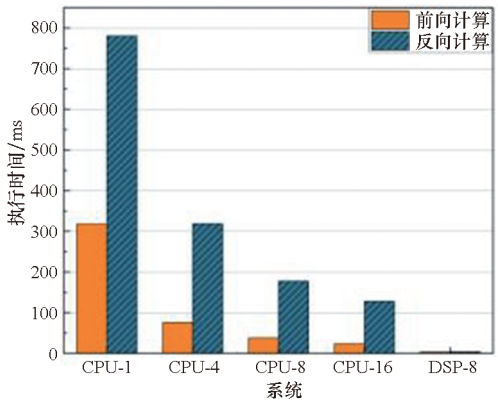
3.1.1　卸载算子计算

在本节对比了 DSP 端算子优化前后性能。优化前实验设置为使用 TensorFlow 算子库在纯 CPU 上进行计算。根据使用核数的不同分别记为 CPU-1、CPU-4、CPU-8、CPU-16。优化后实验使用 8 个 DSP 核进行计算,记为 DSP-8。

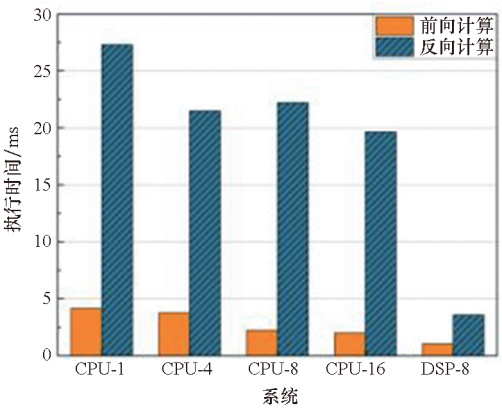
实验结果如图 5 所示。算子优化后各个算子计算速度有不同程度的提升。其中 CONV3_RES 的计算速度提升最为明显。以 CPU-8 作为 Baseline 对比 DSP 端算子计算优化性能。卷积算子的计算速度均有较大提升。CONV3_RES 的一次前向计算耗时从 38.38 ms 降至 4.02 ms,反向计算耗时从 178.05 ms 降至 4.2 ms。加速比分别为 9.54 和 42.29。CONV3_INIT 的前后向加速比分别是 2.31 和 11.53。CONV1_POLICY 的前后向加速比分别是 6.19 和 11.53。CONV1_VALUE 的前后向加速比分别是 1.94 和 9.63。FC 性能提升不理想。原因在于 MiniGo 中 FC2 维度尺寸小,在 DSP 上计算快,申请内存和数据搬运操作相对慢。



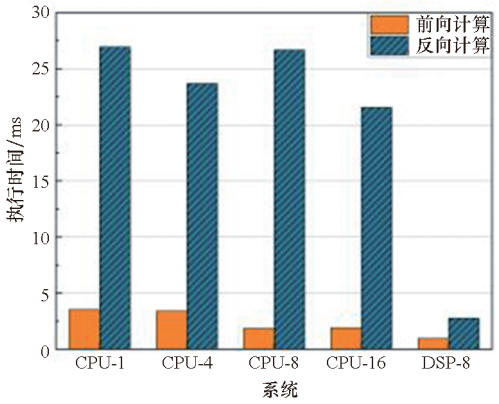
(a) CONV3_INIT



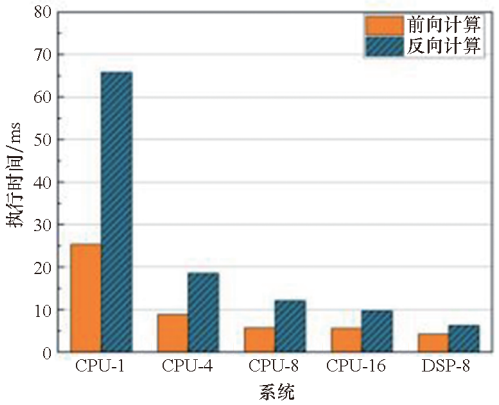
(b) CONV3_RES



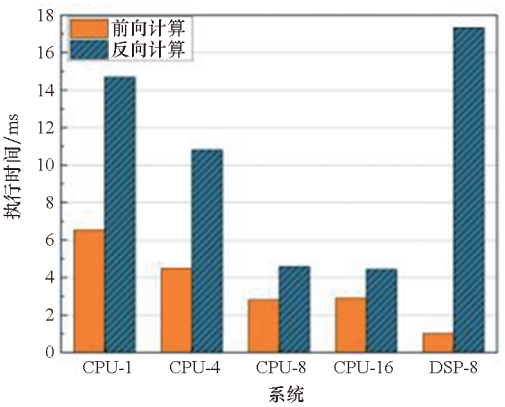
(c) CONV1_POLICY



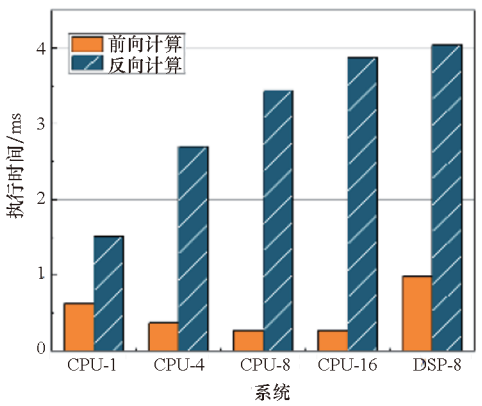
(d) CONV1_VALUE



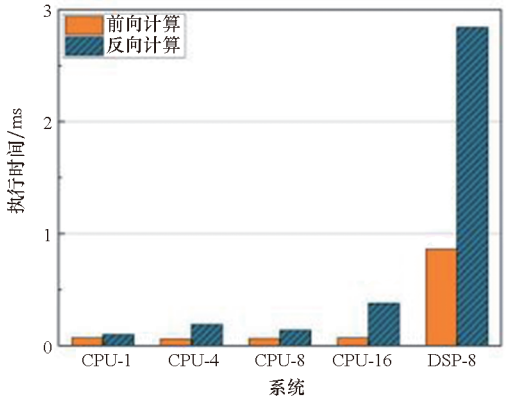
(e) BN + ReLU



(f) FC1



(g) FC2



(h) FC3

图5 各算子在不同CPU核数以及DSP上的计算时间对比

Fig. 5 Comparison of computing time of each operator on different CPU cores and DSP

3.1.2 扩展性

扩展性是验证并行计算方法性能的有效方法。扩展性为固定计算规模、增加计算资源得到的并行效率。在本节讨论了算子计算优化后的扩展性。实验设置为,固定使用的DSP核数为8,输入数据的批大小,分别在数据批大小为96和768上进行训练,记录1步中各算子耗时。重复进行多组实验。该实验以在单核计算的算子计算耗时为基准,记录在8核计算的算子计算耗时相对于基准的加速比。实验结果显示,CONV3_RES算子多核并行后加速比提升最大。当批大小分别是96和768时,CONV3_RES的加速比分别是16.22和24.69。CONV3_INIT的加速比分别是4.40和4.44。BN + ReLU也有明显的速度提升。FC3加速比提升最小。原因是MiniGo中FC算子计算量小。受限于带宽,内存申请、数据传输时间大于计算时间,导致收益不高。实验结果证明,实现算子的多核计算能有效提升计算速度。当一次计算数据规模扩大8倍时,能保持大约一致的加速比。

3.1.3 计算效率

为了验证所提方法在自研原型系统上的有效性,以计算量最大的CONV3_RES为例,分析了其计算量、计算性能和计算效率,如表2所示。

计算量评估公式为 $\frac{2\times N\times W\times H\times 3\times 3\times C\times K}{1\,000\times 1\,000\times 1\,000}$(GFLOPS),其中$N$、$W$、$H$、$C$和$K$定义如前,3×3是CONV3_RES卷积核空间大小。计算性能等于计算量除以平均执行时间。计算效率等于计算性能除以峰值性能。

表 2　DSP-8 下的 CONV3_RES 的计算量、计算性能、计算效率

Tab. 2　Computational efficiency, computational quantity, computational performance of CONV3_RES running on 8 DSP cores

批大小	计算量/GFLOPS	计算性能/(TFLOPS/s)	计算效率/%
96	2.56	0.64	23.03
768	20.44	1.30	46.90

由表 2 结果可知,在相对小的计算量的情况下,计算效率达到 23% 左右。主要原因是计算量小,计算时间相对于数据传输时间占比降低。随着计算量增加,本文实现的计算性能和效率在提升。说明本文方法有效利用了硬件结构特点,能够有效发挥其计算性能。

3.2　异构设备间的共享内存编码优化

本节的基准测试记为 DSP-SMP。使用 TensorFlow 框架的默认异构计算接口,数据在 CPU 和 DSP 之间显式搬运,在 DSP 上进行 MiniGo 训练。优化后实验记为 DSP + SMP,使用共享内存编码优化后,在 DSP 上进行 MiniGo 训练。采用算子计算时间来衡量共享内存编程优化带来的性能提升。

表 3 和表 4 展示了共享内存编码优化前后各个算子计算时间对比。各算子在前向和反向计算中,共享内存编码均表现出一定的性能提升。其中,卷积计算的速度提升最为明显。CONV3_INIT 反向计算的加速比是 11.88。CONV3_RES 前向计算耗时从 16.85 ms 降到 4.02 ms,加速比是 4.19。相应地,反向计算平均耗时从 35.59 ms 降到 4.21 ms,加速比是 8.45。BN + ReLU 前向计算和后向计算的加速比分别是 4.42 和 5.56。从实验结果可知,共享内存编码大大地缩减了计算时间。

表 3　共享内存编码优化前后前向计算平均时间对比

Tab. 3　Comparison of forward calculation time before and after shared memory programming optimization

算子类型	DSP-SMP/ms	DSP + SMP/ms	加速比
CONV3_INIT	16.05	6.52	2.46
CONV3_RES	16.85	4.02	4.19
CONV1_POLICY	18.22	1.05	17.40
CONV1_VALUE	15.95	0.95	16.72
BN + ReLU	18.70	4.23	4.42
FC1 ~ 3	20.98	2.87	7.32

表 4　共享内存编码优化前后反向计算平均时间对比

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

算子类型	DSP-SMP/ms	DSP + SMP/ms	加速比
CONV3_INIT	16.48	1.39	11.88
CONV3_RES	35.59	4.21	8.45
CONV1_POLICY	38.55	3.59	10.75
CONV1_VALUE	30.96	2.77	11.18
BN + ReLU	34.90	6.28	5.56
FC1 ~ 3	27.13	24.21	1.12

3.3　耗时与加速比

在本节中,展示了本文方法在训练阶段和自博弈阶段的整体加速效果。Baseline 实验为 CPU-8,设置见 3.1.1 节。

智能体训练阶段和自博弈阶段耗时对比如表 5和表 6 所示。在训练阶段中,1 步中前向计算总耗时从 606.18 ms 降至 236.79 ms,反向计算总耗时从 1 642.00 ms 降至 350.62 ms。加速比分别为 2.56 和 4.68。1 步训练时间从2 248.18 ms降为 587.41 ms,加速比为 3.83。由于 1 次对局中行动长度波动较大,在自博弈阶段选取前瞻次数为 10 000进行分析。在 CPU-8 中执行 10 000 次前瞻的自博弈推理时间为1 185.87 s,在 DSP-8 中为 816.48 s。DSP-8 在自博弈推理阶段的加速比达 1.5。综上,本文方法能够实现 MiniGo 算子加速。

表 5　1 步内训练耗时对比

Tab. 5　Execution time of one step computation

阶段	CPU-8/ms	DSP-8/ms	加速比
前向计算	606.18	236.79	2.56
反向计算	1 642.00	350.62	4.68
1 步	2 248.18	587.41	3.83

表 6　自博弈推理耗时对比

Tab. 6　Inference execution time of self-play

系统	前瞻次数	时间/s	加速比
CPU-8	10 000	1 185.87	1.0
DSP-8	10 000	816.48	1.5

4　结论

本文结合 MiniGo 计算过程特点和高性能异

构加速器特性,主要针对传统算子加速方法没有实现定制化汇编导致的资源浪费、片上数据传输消耗大和异构计算资源分配的问题进行了优化。首先,针对算子计算 - 访存特性和计算平台资源定制了算子计算,在多核、多 VPE、指令集方面实现了并行计算。然后,结合计算平台拥有设备间共享存储段的特性,设计了共享内存编码模式,减少数据传输消耗。最后,结合算法计算流程在 CPU-DSP 间实现了流水并行,合理分配任务和资源。同时,面向 TensorFlow 实现了一个易于使用的高性能算子库。实验部分对比了本文方法的算子计算在多核并行化的性能提升、共享内存编码模式带来的性能提升。本文方法实现了面向高性能异构加速器的 MiniGo 算子加速。

参考文献(References)

[1] MNIH V, KAVUKCUOGLU K, SILVER D, et al. Playing Atari with deep reinforcement learning[EB/OL]. (2013 - 12 - 19)[2022 - 12 - 01]. https://arxiv.org/abs/1312.5602.pdf.

[2] 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.

[3] ARULKUMARAN K, CULLY A, TOGELIUS J. AlphaStar: an evolutionary computation perspective[C]//Proceedings of the Genetic and Evolutionary Computation Conference Companion, 2019: 314 - 315.

[4] BERNER C, BROCKMAN G, CHAN B, et al. Dota 2 with large scale deep reinforcement learning[EB/OL]. (2019 - 12 - 13)[2022 - 12 - 01]. https://arxiv.org/abs/1912.06680.pdf.

[5] JUMPER J, EVANS R, PRITZEL A, et al. Highly accurate protein structure prediction with AlphaFold[J]. Nature, 2021, 596(7873): 583 - 589.

[6] ZHAO E M, YAN R Y, LI J Q, et al. AlphaHoldem: high-performance artificial intelligence for heads-up no-limit poker via end-to-end reinforcement learning[J]. Proceedings of the AAAI Conference on Artificial Intelligence, 2022, 36(4): 4689 - 4697.

[7] 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.

[8] TROMP J. The number of legal Go positions[EB/OL]. (2016 - 01 - 20)[2023 - 11 - 22]. https://tromp.github.io/go/legal.html.

[9] SILVER D, SCHRITTWIESER J, SIMONYAN K, et al. Mastering the game of Go without human knowledge[J]. Nature, 2017, 550(7676): 354 - 359.

[10] YIN S F, WANG Q L, HAO R C, et al. Optimizing irregular-shaped matrix-matrix multiplication on multi-core DSPs[C]//Proceedings of 2022 IEEE International Conference on Cluster Computing, 2022: 451 - 461.

[11] CHEN Y H, EMER J, SZE V. Eyeriss: a spatial architecture for energy-efficient dataflow for convolutional neural networks[C]//Proceedings of 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), 2016: 367 - 379.

[12] HAN S, LIU X Y, MAO H J, et al. EIE: efficient inference engine on compressed deep neural network[C]//Proceedings of 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), 2016: 243 - 254.

[13] ZENG K, MA Q, WU J W, et al. FPGA-based accelerator for object detection: a comprehensive survey[J]. The Journal of Supercomputing, 2022, 78(12): 14096 - 14136.

[14] FANG J, MULDER Y T B, HIDDERS J, et al. In-memory database acceleration on FPGAs: a survey[J]. The VLDB Journal, 2020, 29(1): 33 - 59.

[15] ABU TALIB M, MAJZOUB S, NASIR Q, et al. A systematic literature review on hardware implementation of artificial intelligence algorithms[J]. The Journal of Supercomputing, 2021, 77(2): 1897 - 1938.

[16] SUI Y, YIN M, XIE Y, et al. CHIP: channel independence-based pruning for compact neural networks[C]//Proceedings of 35th Conference on Neural Information Processing Systems, 2021.

[17] ZHAO J, DI P. Optimizing the memory hierarchy by compositing automatic transformations on computations and data[C]//Proceedings of 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture, 2020: 427 - 441.

[18] JOUPPI N P, YOUNG C, PATIL N, et al. In-datacenter performance analysis of a tensor processing unit[C]//Proceedings of the 44th International Symposium on Computer Architecture (ISCA), 2017.

[19] CHEN Y. A survey on industrial information integration 2016-2019[J]. Journal of Industrial Integration and Management, 2020, 5(1): 33 - 163.

[20] JIANG B L, CHENG X W, TANG S H, et al. APCNN: explore multi-layer cooperation for CNN optimization and acceleration on FPGA[C]//Proceedings of FPGA '21: The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2021: 146 - 147.

[21] HUANG X D, WANG Q L, LU S Y, et al. Evaluating FFT-based algorithms for strided convolutions on ARMv8 architectures[J]. Performance Evaluation, 2021, 152: 102248.

[22] HUANG X D, WANG Q L, LU S Y, et al. NUMA-aware FFT-based convolution on ARMv8 many-core CPUs[C]//Proceedings of 2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking(ISPA/BDCloud/SocialCom/SustainCom), 2021: 1019 - 1026.

(编辑:梁慧,杨琴)