计算机科学
COMPUTER SCIENCE

基于国产众核处理器的深度神经网络算子加速库优化

高捷，刘沙，黄则强，郑天宇，刘鑫，漆锋滨

引用本文

高捷，刘沙，黄则强，郑天宇，刘鑫，漆锋滨. 基于国产众核处理器的深度神经网络算子加速库优化[J]. 计算机科学，2022, 49(5): 355-362.

GAO Jie, LIU Sha, HUANG Ze-qiang, ZHENG Tian-yu, LIU Xin, QI Feng-bin. Deep Neural Network Operator Acceleration Library Optimization Based on Domestic Many-core Processor[J]. Computer Science, 2022, 49(5): 355-362.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.210500226

基于国产众核处理器的深度神经网络算子加速库优化

高 捷[1] 刘 沙[2] 黄则强[2] 郑天宇[3] 刘 鑫[2] 漆锋滨[2]
1 信息工程大学网络空间安全学院 郑州 450000
2 江南计算技术研究所 江苏 无锡 214083
3 山东大学软件学院 济南 250101
(george_jie_work@163.com)

摘 要 基于不同硬件设备的算子加速库已经成为深度学习框架不可或缺的一部分,能够为大规模训练或者推理任务提供数倍的性能加速。当前的主流算子库都是基于GPU架构开发的,与其他异构设计并不兼容;SWDNN算子库是基于申威26010开发的,无法充分发挥升级后的申威26010 pro处理器的性能,也不能满足当前GPT-3等大型神经网络模型对大容量内存和高访存带宽的需求。文中面向申威26010 pro处理器体系结构的特点和大型神经网络模型的训练需求,提出了基于多核组的三级并行和神经网络算子任务调度方案,在满足大型模型训练内存需求的同时,提高了并行效率和整体计算性能;提出了三级异步流水机制和计算访存重叠的访存优化方法,显著缓解了神经网络算子的访存性能瓶颈。基于以上方法,文中构建了基于申威26010 pro处理器的SWTensor多核组算子加速库,在自然语言处理模型GPT-2上进行了实验,结果表明,其典型计算密集型算子和访存密集型算子在单精度浮点计算性能和访存带宽上分别达到了理论峰值的90.4%和88.7%。

关键词:深度神经网络;算子加速库;负载均衡;异步流水;双缓冲
中图法分类号 TP311

Deep Neural Network Operator Acceleration Library Optimization Based on Domestic Many-core Processor

GAO Jie[1],LIU Sha[2],HUANG Ze-qiang[2],ZHENG Tian-yu[3],LIU Xin[2] and QI Feng-bin[2]
1 Department of Cyberspace Security Academy,Information Engineering University,Zhengzhou 450000,China
2 Jiangnan Institute of Computing Technology,Wuxi,Jiangsu 214083,China
3 School of Software,Shangdong University,Jinan 250101,China

Abstract Operator acceleration libraries based on different hardware devices have become an indispensable part of deep learning framework,which can provide performance improvement for large-scale training or inference tasks dramatically. The current mainstream operator libraries are all developed based on GPU architecture,which is not compatible with other heterogeneous designs. SWDNN operator library is based on the development of SW26010 processor,which can not give full play to the performance of the upgraded SW26010 pro processor,nor can it meet the needs of the current large neural network models such as GPT-3 for large memory capacity and high memory access bandwidth. According to the architecture characteristics of SW26010 pro processor and the training requirements of large neural network model,a three-level parallel and neural network operator task scheduling scheme based on multi-core group is proposed,which can satisfy the memory requirements of large model training and improve the overall computing performance and parallel efficiency. A memory access optimization method with triple asynchronous flow and overlap of computation and memory access is proposed,which significantly alleviates the memory access performance bottleneck of neural network operators. Based on the above methods,this paper constructs the SWTensor many-core group operator acceleration library based on the SW26010 pro processor. The experimental results of natural language processing model GPT-2 show that,computation-intensive operators and memory access intensive operators in SWTensor operator library reach the maximum of 90.4% and 88.7% of the theoretical peak values respectively in single-precision floating-point computing performance and memory access bandwidth.

Keywords Deep neural network,Operator acceleration library,Load balancing,Asynchronous flow,Double-buffering

1 引言

深度神经网络的概念自20世纪80年代提出以来,已经在计算机视觉[1]、自然语言处理[2]、推荐系统[3]等诸多领域被证明其具有高效性。近年来,随着相关技术的不断发展,深度学习技术已经被应用到无人驾驶汽车研究[4]、人工智能体

到稿日期:2021-05-31 返修日期:2021-06-27
基金项目:国家自然科学基金(U1806205)
This work was supported by the National Natural Science Foundation of China(U1806205).
通信作者:刘鑫(yyylx@263.net)

开发[5]、分子结构模拟[6]等更加复杂的场景中，但是这也导致了深度神经网络的模型大小和复杂度不断增加，从而需要更多的算力和内存容量来完成训练。

为了缩短训练时间，在高性能计算机上进行深度神经网络的研究成为了一种有效的方式。由于充分利用高性能计算机性能的需要，针对特定硬件的架构设计相应的算子加速库也成为一种通用的解决方案。当前在商用领域以及科研领域，英伟达提供的 GPU 几乎是用户们的唯一选择，针对 GPU 的架构，英伟达设计的 cuDNN[7]库提供了简洁易用的 API，并将其广泛用于 Caffe[8]，Tensorflow[9]，PyTorch[10]等热门深度学习框架。其他的 GPU 加速库如 maxDNN[11]，Caffe con Troll (CcT)[12]，fbfft[13]等也是基于特定的算法或者网络而设计的，在某些情况下可以获得比 cuDNN 更好的性能。

而我国自主研发的超级计算机“神威太湖之光”[14]的硬件架构迥异于 GPU，采用了我国独立研制的申威系列众核处理器，如申威 26010 以及迭代更新后的申威 26010 pro。然而，cuDNN 等高效的深度学习算子库并不支持申威众核处理器的架构，且直接运行深度学习任务对神威太湖之光的计算性能的利用率不足 1%。

SWDNN[15]算子库是少有的基于申威 26010 开发的算子库。其通过执行如卷积循环的组织排列、阻塞技术、寄存器数据通信方案以及两条指令流水线的重新排序策略，成功实现了 1.6 TFLOPS 的双精度卷积计算性能，最高可达到理论峰值的 54%。

遗憾的是，基于申威 26010 众核处理器的结构自主研发的 SWDNN 算子库无法充分挖掘更新后的申威 26010 pro 处理器的计算性能。

除了计算性能，大规模深度学习模型对于内存大小和访存带宽也有严苛的要求，申威 26010 pro 处理器能够切换相应的模式来满足要求，但这种模式无法得到现有 SWDNN 算子库的支持。

针对上述挑战，本文在对深度学习常见算子和申威 26010 pro 结构进行分析的基础上构建了名为 SWTensor 的多核组算子加速库。

本文第 2 节介绍了当前深度学习训练所面临的挑战，以及新一代申威 26010 pro 众核处理器的架构；第 3 节和第 4 节分别讲述了本文研究的 SWTensor 多核组算子加速库在并行优化和访存优化方面的工作；第 5 节针对上述工作进行了实验验证和性能评估；最后总结全文。

2 背景

2.1 训练面临的挑战

海量的数据集以及更大更复杂的模型网络是近年来深度学习训练中面临的挑战。

由于大数据时代的到来，数据规模在近年来呈爆发式增长，在科研领域，2019—2020 年产生的数据占过去数据的 90%以上。举例来说，新冠病毒的图形分析数据达到了 393 TB；根据欧洲中期天气预报中心(ECMWF)的报告，气象预测的数据量估计每天高达 280 TB；在天文领域，平方公里射电阵(SKA)检测的数据量达到了每秒 16 TB；而美国航空航天局(NASA)模拟探测器登陆火星的数据量为 550 TB。这种规模的数据所构成的数据集往往需要更为复杂的模型来提取特征。

此外，自从基于 Transformer[16]的一系列模型被提出后，深度学习模型的大小每年以指数级的速度增长。2018 年 6 月，由 12 层 transformer 模块组成的 GPT[17]模型是第一个参数量达到亿级的模型；同年 10 月，BERT[18]模型将 transformer 模块的数量翻倍，使参数量达到 3 亿左右；2019 年 2 月，GPT-2[19]模型继续堆叠 transformer 模块到 48 个，拓展参数量到 15 亿，而基于 GPT-2 改进的 Megatron-LM[20]和 Tu-ring-NLG[21]分别将这一数字增加到了 83 亿和 170 亿；2020 年 3 月，GPT 系列的下一个产品 GPT-3[22]继续延续了自己的单向语言模型训练方式，将模型参数放大到了 1 750 亿；2021 年 1 月初，谷歌大脑提出了一个名为 Switch Transformer[23]的简化稀疏模型架构，将自然语言处理模型的参数量扩展至 1.6 万亿。如图 1 所示，从 2018—2021 年期间，深度学习模型规模增加了 4 个数量级。

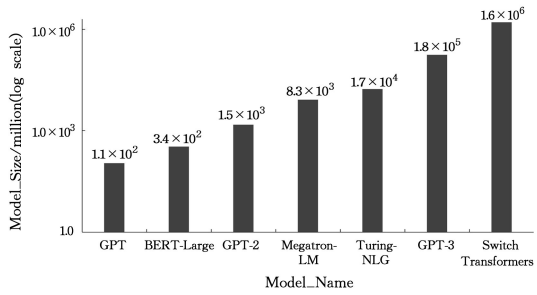

图 1 Transformer 系列模型的规模变化

Fig. 1 Scale changes to the model based on Transformer

2.2 申威 26010 pro 众核处理器

2015 年，国家 863 计划支持的“神威太湖之光”落户无锡，该系统是我国首台全部采用国产处理器构建的世界第一的超级计算机。2016 年 6 月至 2017 年 11 月，在世界超级计算机排名榜单 TOP500 上，“神威太湖之光”连续 4 次登顶。

此后，“神威太湖之光”进行了数次迭代升级，目前采用最新的自主研发的 SW26010 pro 众核处理器，其组成结构如图 2 所示。该申威系列处理器采用异构众核架构，由 6 个运算核组构成，每个核组包括 65 个核心：1 个运算控制核心(MPE)以及由 64 个运算核心(CPE)组成的运算核心阵列(CPEs)，运算核心阵列以 8×8 网格的方式构成。

运算控制核心和运算核心都是完整的 64 位 RISC 内核，但在计算过程中起着不同的作用。运算控制核心支持完整的中断、内存管理和无序问题的执行功能，擅长处理管理、任务调度和数据通信等任务。运算核心旨在最大化地聚合计算吞吐量，实现对核心功能模块的加速。

每个运算核组通过内存控制器(MC)连接到自己的16 GB DDR4 内存，由运算控制核心和运算核心阵列共享。片上网络(NoC)连接全部(6 个)核组与系统接口，6 个核组的存储器也通过 NoC 连接。用户可以显式地设置每个核组的私有内存空间的大小，以及 6 个核组之间共享的内存空间大小。

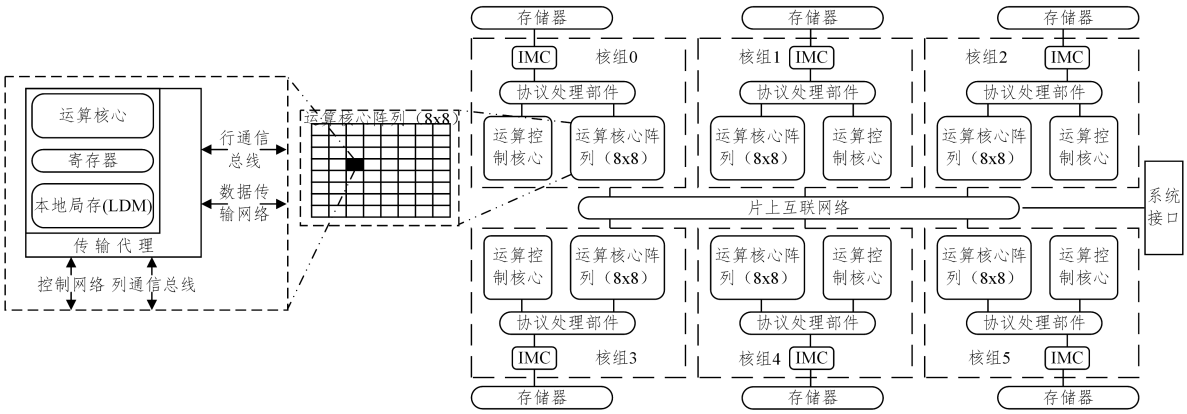

图 2　申威众核处理器 SW26010 pro 架构图

Fig. 2　General architecture of SW many-core processor SW26010 pro

3　并行优化

3.1　三级并行

为了满足大规模深度学习模型对内存的需求，必须考虑在申威 26010 pro 上启用大共享多核组模式，合并多个核组的内存空间。

得益于“神威太湖之光”的运算核组-运算核心阵列-运算核心的结构特点，SWTensor 多核组算子加速库可以实现核组并行、线程并行、SIMD 数据并行 3 个不同层次的并行计算加速。第一级核组并行可以理解为核组间的线程组并行，即程序可以启动同一处理器上的多个运算核组来并行完成任务，通过运算核组中运算控制核心上运行的程序来分配任务的计算负载。第二级线程并行主要指通过运算核组中至多 64 个运算核心上运行的线程来并行完成程序的加速任务。第三级 SIMD 数据并行指单个运算核心上的计算操作可以通过相关 SIMD 编程接口来实现数据处理的矢量化，以充分利用申威众核芯片的矢量指令加速能力。

对于一般的深度学习课题来说，使用线程并行、SIMD 数据并行两级并行即可，也就是在多进程任务中，每个任务进程仅仅启动处理器的一个核组，运算控制核心完成不可众核并行部分的计算以及通信，运算核心阵列进行并行加速。而对于大规模的模型训练任务，每个进程仅仅启动处理器的单个核组往往会导致核组的内存捉襟见底，因此我们考虑用一个进程来启动一块处理器中的全部核组，对每个核组的内存进行重新编址，使其在逻辑上合并成一个更大的内存，并对每个核组可见。这种全共享模式的核组并行能将原本的多个进程的任务用一个进程实现，大大提高了并行效率，解决了内存有限的问题。

3.2　负载均衡

一个申威处理器最多能实现 6 核组的并行，理论上相比单核组能达到 6 倍的计算性能加速。为了保证三级并行的加速效果，需要维持基于三级并行层面的负载均衡，因此设计了 SWTensor 多核组算子加速库的任务调度方案。对于一个深度学习中的算子问题，假设使用 N 个运算核组进行计算，每个核组采用全部的 64 个运算核心，任务调度方案则需要考虑尽量均衡地将数据分为 $N*64$ 份，以便将其映射到运算核心阵列上。深度学习任务中的数据通常是多维数组，对于这样的数据切分，需考虑其切分的维度和数据量，数据切分方案将遵循以下策略：1）避免增加额外的通信；2）避免增加额外的计算；3）基于数据量较大的维度进行切分；4）尽量保证切分的维度刚好能被整除。

这里以典型计算密集型算子三维矩阵乘 baddbmm 算子为例，如图 3 所示，其任务调度方案的描述如算法 1 所示。

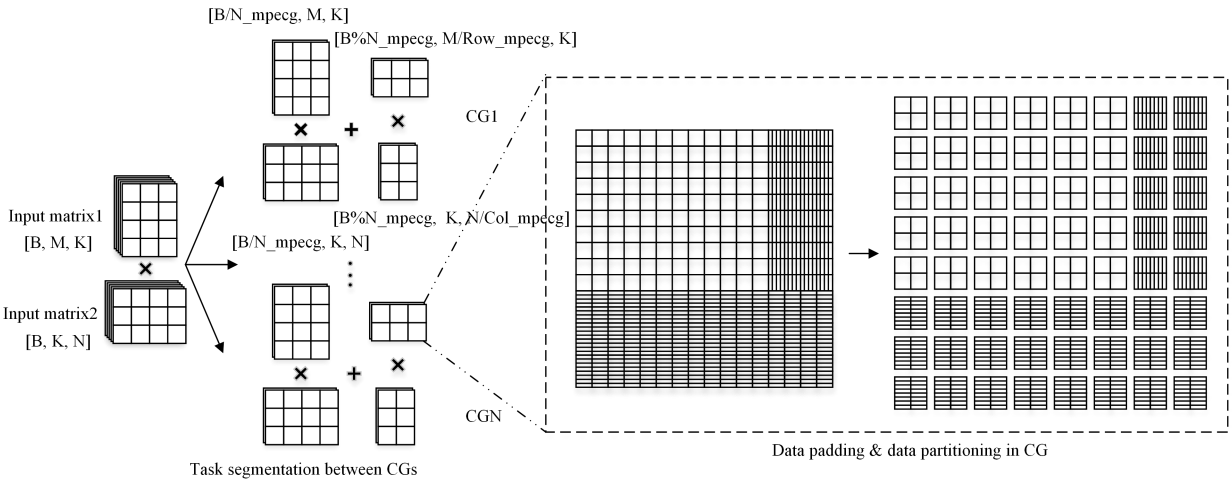

图 3　baddbmm 算子基于负载均衡的任务调度方案

Fig. 3　Task scheduling scheme of baddbmm operator based on load balancing

算法 1 baddbmm 算子的任务调度方案

输入：矩阵 input1[B,M,K] 和 input2[B,K,N]，常数参数 α 和 β；核组数 N_mpecg；行切分数 Row_mpecg；列切分数 Col_mpecg

（保证 Row_mpecg * Col_mpecg ≤ N_mpecg）

输出：矩阵 output[B,M,N]

1. 将输入矩阵 input1[B,M,K]和 input2[B,K,N]按最高维度 B 维度进行均匀切分，平分给每个核组，即每个核组分得的输入数据形状为 input1_i[B/N_mepcg,M,K]，input2_i[B/N_mepcg,K,N]；对于无法除尽的情况，将剩余的数据进行更低维度的细粒度切分，为了避免额外的加法，对矩阵 input1 进行行切分，对矩阵 input2 进行列切分，此时每个核组获得剩余的输入数据形状为 input1_i[B % N_mepcg,M/Row_mpecg,K]，input2_i[B % N_mepcg,K,N/Col_mpecg]。
2. 在每个核组内部的运算核心阵列上，对于获得的输入数据，首先要用 0 填充对数据进行扩展，使其在行和列上都为 8 的倍数，然后将数据均匀地分为 64 份，每份分配给一个运算核心用于计算。
3. 对于分配给每个运算核心的数据，使用 simd 相关接口继续对数据进行矢量化处理，然后进行并发数学运算。

4 访存优化

单块申威 26010 pro 处理器的计算性能非常高，与 NVIDIA 的 Tesla V100 GPU 接近，然而 Tesla V100 GPU 的总带宽高达 900 GB/s，相比之下，申威 26010 pro 处理器的访存带宽不到 Tesla V100 GPU 的 1/2，因此很容易陷入带宽受限的情况。由于大规模深度学习模型对访存同样有着非同一般的需求，因此无论是计算密集型算子还是访存密集型算子，都需要考虑在访存层面进行优化。

在新一代申威众核处理器中内存和访存方式有一些独特的布局和机制，需要在分析清楚的情况下进行优化。处理器中的内存分为 3 种，即连续段、交叉段和私有段，其中连续段加上交叉段被称为主存。连续段是每个核组的私有空间，运算控制核心与运算核心均可直接访问本核组的连续段空间。交叉段是同一处理器内所有核组间的共享空间，运算核心能够访问全芯片的该部分共享区域。私有段指每个运算核心具有一块高速的本地局部数据存储空间（LDM），运算核心局存空间分为私有空间和连续共享空间，运算核心局存私有空间是本运算核心快速访问的本地私有空间；运算核心局存连续共享空间是运算核心用于阵列内运算核心局存共享访问的空间，运算核心可以访问到其他运算核心的局存空间。

访存通信方面主要有 DMA（Direct Memory Access）和 RMA（Remote Memory Access）两种方式。运算核心可以发起 DMA 操作，以实现运算核心局存与主存之间的数据传输，提高运算核心访存的效率。运算核心可以通过 RMA 操作来实现运算核心局存与其他运算核心局存之间的数据传输，即 RMA 操作能够实现运算核心间的高效通信，可以缓解运算核心局存容量受限的问题。

4.1 基于异步流水的三级存储访问机制

直接访问主存通常会造成严重的访存瓶颈，因此在多核组算子加速库中采用基于异步流水的三级存储访问机制来提高访存带宽。用于计算的数据通过主存-运算核心局存（LDM）-寄存器这 3 层结构进行传递，如图 4 所示，其具体步骤为：第一步，核组中的每个运算核心阵列先根据算法 1 中的任务调度方案获得该运算核心阵列要计算的数据块；第二步，通过 DMA 存取操作将数据块从主存加载到对应的 LDM，通过 RMA 操作将要共享的数据块传递给相应的运算核心；第三步，执行 ld/st 类访问指令，将数据从 LDM 移动到寄存器中，以便随后进行计算。其中，对于无规则的离散数据，DMA 操作访问 LDM 的效率会大大降低，因此使用 cache 访问取代 DMA 操作：首先在运算核心的 LDM 中开辟出一块 cache 空间，对主存的可 cache 空间进行 ld/st 类访问，将数据从主存读取到 LDM 的 cache 空间。

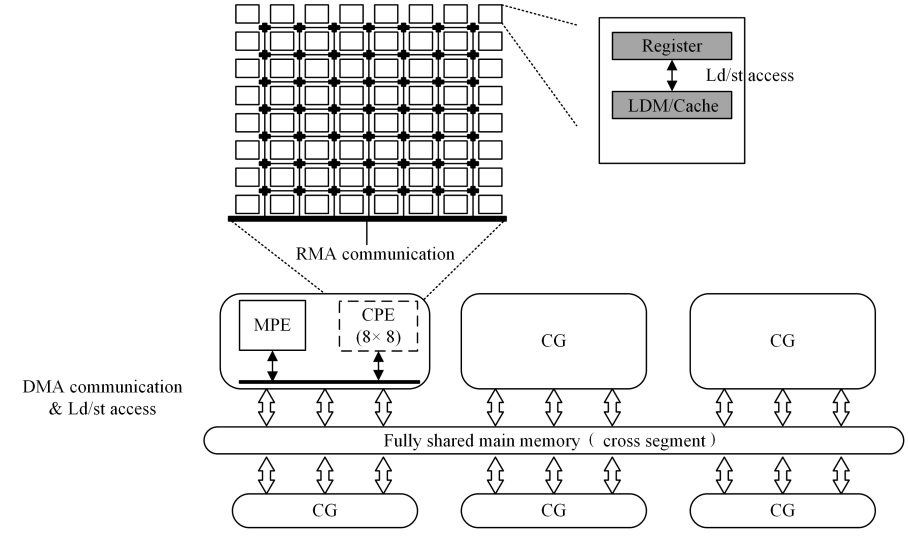

图 4 SWTensor 算子库中的基于异步流水的三级存储访问机制优化

Fig. 4 Three-level storage access mechanism based on asynchronous pipelining in SWTensor operator library

通过对 LDM 和寄存器进行额外的两级控制，多核组算子加速库可以实现有效的数据共享，从而降低从主存中进行

的实际数据访问所需的访存带宽。这样通过异步流水线执行机制,隐蔽了访存带来的额外的性能开销,在保证训练速度的同时,使训练的模型大小不再受制于 LDM 的容量。

4.2　计算和访存重叠机制

为了进一步提高访存效率,多核组算子加速库采用非阻塞通信结合双缓冲的方法来实现 DMA 访存或 RMA 访存与计算的重叠。其思路是:将运算核组的 LDM 缓冲区分为两部分,当数据在一个 LDM 缓冲区中计算时,将下一次迭代需要的数据通过 DMA\RMA 操作加载到另一个 LDM 缓冲区,即实现了计算与访存操作的并发进行。计算和访存重叠机制主要用到两种操作:通过定义的非阻塞读写操作 iget/iput 来实现访存;用 wait_value 接口来判断相应的访存操作是否完成。以 DMA 传输为例,算法的具体步骤如算法 2 所示。

算法 2　计算和访存重叠优化算法

```
输入:主存数据地址 get_MEM1,get_MEM2
输出:主存数据地址 get_MEM1,get_MEM2
DMAiget(get_LDM1,get_MEM1,get_reply_1)
/*通过 DMA 操作将数据从主存地址 get_MEM1 读到 LDM 地址 get_LDM1,通过标志 get_reply_1 判断是否读取完毕*/
DMAiget(get_LDM2,get_MEM2,get_reply_2);
for(int i=0;i < total_size; i += part_size){
    DMAwait_value(put_reply_1,1); /*等待 DMA 操作写回完毕*/
    put_reply_1=0;
    DMAwait_value(get_reply_1,1); /*等待 DMA 操作读取完毕*/
    get_reply_1=0;
    simd_compute(get_LDM1); /*对数据矢量化计算*/
    DMAiput(put_MEM1,put_LDM1,
    put_reply_1); /*写回数据*/
    DMAiget(get_MEM1,get_LDM1,get_reply_1);

    /*第二块 LDM 对不同地址数据执行相同操作*/
    DMAwait_value(put_reply_2,1);
    put_reply_2=0;
    DMAwait_value(get_reply_2,1);
    get_reply_2=0;
    simd_compute(get_LDM2);
    DMAiput(put_MEM2,put_LDM2,put_reply_2);
    DMAiget(get_MEM2,get_LDM2,get_reply_2);
}
```

5　实验结果与性能分析

5.1　实验设计

本文选择基于国产超级计算机"神威太湖之光"和移植于该平台上的深度学习框架 PyTorch 进行实验。以每个 SW26010 pro 众核处理器为一个工作节点,每个节点包括 6 个核组。设计多项实验,来验证多核组算子加速库的性能。

5.2　正确性验证

为了证明 SWTensor 多核组算子加速库在国产超算平台上的正确性,本文将 PyTorch 中的 CPU 版本算子实现结果作为基线参考标准,将相同输入数据下的多核组算子加速库的相应算子结果与基线参考标准的接近程度作为准确率。实验选取了不同规模和形状的数据,对单精度、双精度以及半精度这 3 种数据精度进行了多次测试,统计多核组算子加速库的算子结果在基线参考标准下的平均误差,并用它来表示 SWTensor 算子库的准确率。这里定义准确率为:

$$\text{准确率}=(\text{基线参考标准}-|\text{平均误差}|)/\text{基线参考标准}\times 100\% \tag{1}$$

如表 1 所列,在 3 种常见精度下,SWTensor 算子库的算子结果与 PyTorch 的 CPU 版本实现的结果基本吻合。

表 1　SWTensor 算子库在不同数据精度下的准确率

Table 1　Accuracy of SWTensor operator library in different data precision

数据精度	单精度	双精度	半精度
准确率/%	99.99	99.99	99.87

5.3　性能测试

为了验证 SWTensor 多核组算子加速库在并行优化和访存优化策略上的有效性,本文分别进行了如下实验。

5.3.1　三级并行可扩展性验证

三级并行方法是在原有的线程并行、SIMD 数据并行两级的基础上增加了一级核组并行,将单个进程从单核组拓展到了多核组。因此,需要对这种方法的可扩展性进行验证。本文对多个算子进行了可扩展性实验,这里定义平均效率为:

$$\text{Average\_Efficiency}=\text{Base\_time}/(\text{Run\_time}\times\text{CG\_Number})\times 100\% \tag{2}$$

其中,Base_time 为在单个核组上运行算子得到的基线时间,而 Run_time 则是选择不同数量的核组运行算子得到的时间,CG_Number 为参与运行的核组数量。具体结果如图 5 所示,当核组数扩展到 2 时,由于通信访存开销、多核组启动等额外损耗,计算效率有所下降,下降程度约为 1.6%,并没有达到理想的加速比。但是由于对负载均衡和访存的优化,这些额外损耗并未随着核组数量的增加而成比例上升,因此平均计算效率不断上升,接近理想情况。

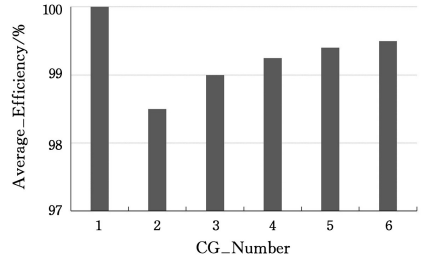

图 5　不同核组数量下 SWTensor 算子库的平均效率

Fig. 5　Average efficiency of SWTensor operator library with different number of CGs

5.3.2　负载均衡验证

本节以计算密集型算子 baddbmm 为例,通过将是否采用

基于负载均衡的任务调度方案的两种结果进行对比，来衡量SWTensor多核组算子加速库的负载均衡能力。具体使用单块处理器，即6个核组来运行该算子，Batch_size为输入矩阵的第一维度。策略1不使用SWTensor的负载均衡任务调度方案，而策略2则采用该方案，两种策略的结果如图6所示。

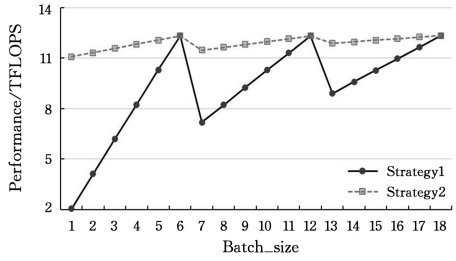

图6 两种策略下baddbmm算子在不同Batch_size中的浮点性能

Fig. 6 Floating point performance of baddbmm operator at different Batch_size under two strategies

从图6可知，策略1使用粗粒度的任务调度方案，仅仅按照Batch_size进行数据切分，导致当Batch_size的值不是6的倍数时存在部分核组处于空闲状态，产生严重负载不均衡，使得平均浮点性能下降。因此，策略1的浮点性能曲线总是以横坐标的6为周期，先下降再上升，并在Batch_size为6的倍数时达到峰值。相比之下，策略2的浮点性能曲线更为平滑，浮点性能仅仅在11.1～12.3 TFLOPS范围内波动，证实了SWTensor负载均衡任务调度方案的有效性。

5.3.3 基于异步流水的三级存储访问机制验证

该三级存储访问机制主要是用异步流水的方式使数据从主存到LDM再到寄存器之间流动，理论上能大大提高访存的效率。首先在访存密集型算子masked_fill上比较三级存储访问与直接访存在不同数据量下的访存带宽差距，如图7所示。直接访存的实际带宽非常低，不到20 GB/s，而采用三级存储访问的方式使处理器带宽的利用率有了较大的提高，基本稳定在200～220 GB/s左右。相比直接访存，基于异步流水的三级存储访问机制的访存利用率提高了63.3%～70.4%。

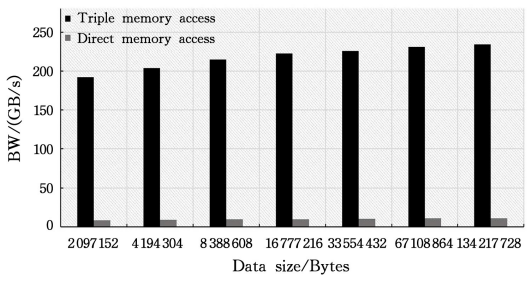

图7 两种策略下masked_fill算子在不同数据量中的访存带宽

Fig. 7 Bandwidth of masked_fill operator in different data quantities under the two strategies

此外，当传输的数据为无规则的离散数据时，采用argmax算子来评估这种情况下的优化结果。在argmax算子中访存得到的数据往往都是少量且离散的，由于LDM的DMA操作更适合传输连续的数据块，argmax算子的实际带宽利用率将会大幅下降。如图7和图8所示，在同样的数据量下，尽管都是基于异步流水的三级存储访问方法读取数据，argmax算子中的实际带宽利用率相比masked_fill算子下降了60.7%～75.2%。因此，用一级cache代替三级存储访问机制中的LDM，用ld/st类访问代替DMA操作。这种措施有效减少了带宽利用率的损失，将带宽稳定到了150～190 GB/s，图8所示的结果证明了其有效性。

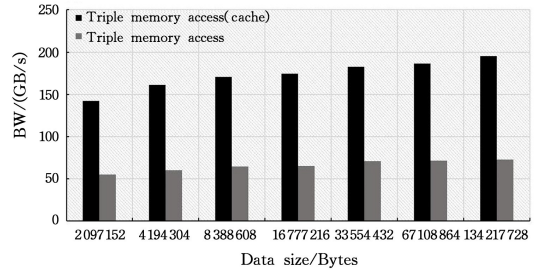

图8 两种策略下argmax算子在不同数据量中的访存带宽

Fig. 8 Bandwidth of argmax operator in different data quantities under the two strategies

5.3.4 计算和访存重叠机制验证

计算和访存重叠机制适用于计算与访存并重的算子，通过牺牲部分LDM空间、增加DMA传输次数来实现将访存开销隐藏在计算时间中。如图9中的柱状图所示，该方法将单次计算和访存分为两次完成，利用非阻塞通信将访存操作与下一次计算同步进行，理想情况下能够将访存开销降低到可接受的范围，其不会随着数据量的增大而线性变大。这里以softmax算子进行验证，如图9中的折线图所示，计算和访存重叠机制的运行时间比正常方法有所缩短，缩短的时间随着数据量的增大而增长，最高能有35.7%的性能提高，改善了访存瓶颈对计算与访存并重型算子的性能影响。

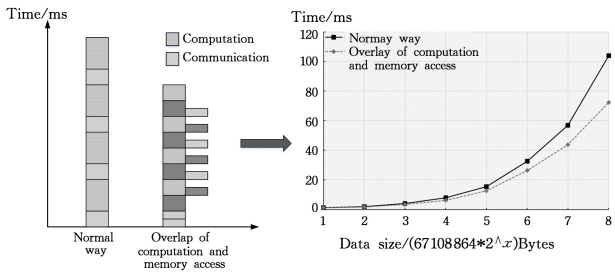

图9 计算和访存重叠机制下softmax算子的时间曲线

Fig. 9 Time curve of softmax operator under calculation and access overlap mechanism

5.3.5 总体性能分析

本文采用自然语言处理模型GPT-2来评估SWTensor算子库的整体性能。实验采用了8块SW26010 pro处理器，使用PyTorch深度学习框架，依赖SWTensor算子库；对比组采用了8块Tesla V100 GPU，同样使用PyTorch深度学习

框架，并采用 cuDNN v7.6.5 加速库。

由于 SW26010 pro 与 Tesla V100 GPU 的各项参数与性能并不相同，因此无法通过运行时间来直接衡量两种算子加速库的性能。如表 2 所列，本文通过运行中的 GPT-2 模型的峰值浮点性能占比与峰值带宽占比来对比上述两种算子库的性能。尽管从单步迭代时间来看，GPT-2 调用 SWTensor 算子库的时间为 662.1 ms，是调用 cuDNN 的时间的 2.6 倍，但是在 SWTensor 算子库中的计算密集型算子对 SW26010 pro 的峰值浮点性能利用率最高达到了 90.4%，访存密集型算子对峰值访存带宽利用率最高达到了 88.7%，同 cuDNN 在 GPU 上的利用率较为接近，只有约 2%～3%的差距，其性能远超过先前开发的 SWDNN 算子库，从这方面来说，SWTensor 算子库的性能是可以接受的。

表 2　GPT-2 在两种处理器上的整体性能

Table 2　Overall performance of the GPT-2 on both processors

	SW26010 pro	V100
峰值浮点性能占比/%	90.4	92.60
峰值带宽占比/%	88.7	89.40
单步迭代时间/ms	662.1	254.3

为了进一步评估两种处理器性能对模型训练产生的影响，实验选取了在 GPT-2 单步迭代中耗时排名前 11 的算子，对比了它们在不同算子加速库上运行的结果。在单精度浮点性能方面 SW26010 pro 与 Tesla V100 GPU 较为接近，而在访存带宽方面 SW26010 pro 则与 V100 相差很大，具体如图 10 所示。在以 gemm 为主的计算密集型算子中，SWTensor 算子库在时间上与 cuDNN 加速库同样较为接近，约慢 10%～12%；而在 masked_fill 等访存密集型算子的运行时间上，与 cuDNN 相比，SWTensor 算子库最多有 60%的性能下降。整体时间上，如表 2 所列，从单步迭代时间上来看，GPT-2 模型调用 SW26010 pro＋SWTensor 的时间是调用 Tesla V100 GPU＋cuDNN 的 2.6 倍左右。

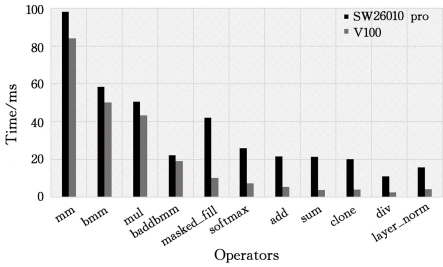

图 10　GPT-2 中主要算子在两种处理器上的运行时间

Fig. 10　Elapsed time of the major operators in GPT-2 on both processors

结束语　本文提出了基于国产申威众核处理器 26010 pro 设计的多核组算子加速库 SWTensor，旨在提高深度学习任务中对处理器计算性能和访存带宽的利用率。基于对申威 26010 pro 架构的分析，本文采用了一系列优化方案：三级并行以及负载均衡任务调度方案保证了多核组算子相比单核组最高提升 6 倍的并行性能，并维持了核组间的负载均衡；访存优化中基于异步流水的三级存储访问机制为访存利用率带来了 63.3%～70.4%的提升，而计算和访存重叠机制则在一定条件下为部分算子的访存利用率带来了 35.7%的提升。最终 SWTensor 在单精度浮点计算性能上，以 gemm 为主的计算密集型算子最高达到了理论峰值的 90.4%；在访存带宽上，以 mask_fill 为主的访存密集型算子最高达到了理论峰值的 88.7%。SWTensor 算子库在主流框架和模型中能够正常运行，具有较好的通用性和可推广性。然而，对比实验结果证明，SWTensor 算子库与 cuDNN 算子库相比还具有一定的差距。因此，下一步工作为：一方面继续优化现有算子，提高对国产申威处理器的性能利用率，另一方面研究其他常用算子，丰富 SWTensor 算子库的算子种类，使其对深度学习任务的支持更为全面。

参 考 文 献

[1] HUANG G，LIU Z，LAURENS V，et al. Densely Connected Convolutional Networks[C]// IEEE Computer Society. 2016.

[2] BROWN T B，MANN B，RYDER N，et al. Language Models are Few-Shot Learners[C]// Conference and Workshop on Neural Information Processing Systems. 2020.

[3] NAUMOV M，MUDIGERE D，SHI H，et al. Deep Learning Recommendation Model for Personalization and Recommendation Systems[J]. arXiv：1906.00091，2019.

[4] KOSSMANN J，SCHLOSSER R. Self-Driving Database Systems：A Conceptual Approach[J]. Distributed and Parallel Databases，2020，38：1-2.

[5] YE D，CHEN G，ZHANG W，et al. Towards Playing Full MOBA Games with Deep Reinforcement Learning[C]// Conference and Workshop on Neural Information Processing Systems. 2020.

[6] HEO L，FEIG M. High-Accuracy Protein Structures By Combining Machine-Learning With Physics-Based Refinement[J]. Proteins-Structure Function and Bioinformatics，2020，5：637-642.

[7] CHETLUR S，WOOLLEY C，VANDERMERSCH P，et al. cuDNN：Efficient Primitives for Deep Lear-ning[C]// Deep Learning and Representation Learning Workshop (NIPS2014). 2014.

[8] JIA Y，SHELHAMER E，DONAHUE J，et al. Caffe：Convolutional Architecture for Fast Feature Embedding[C]// Proceedings of the 22nd ACM International Conference (MM'14). 2014.

[9] ABADI M，AGARWAL A，BARHAM P，et al. Tensor Flow：Large-Scale Machine Learning on Heterogeneous Distributed Systems[J]. arXiv：1603.04467，2015.

[10] PASZKE A，GROSS S，MASSA F，et al. PyTorch：An Imperative Style，High-Performance Deep Learning Library[J]. arXiv：1912.01703，2019.

[11] LAVIN A. maxDNN：An Efficient Convolution Kernel for Deep Learning with Maxwell GPUs[J]. arXiv：1501.06633，2015.

[12] STEFAN H，FIRAS A，CE Z，et al. Caffe con troll：Shallow

ideas to speed up deep learning[C]// Proceedings of the Fourth Workshop on Data Analytics in the Cloud. 2015.

[13] VASILACHE N, JOHNSON J, MATHIEU M, et al. Fast Convolutional Nets With fbfft: A GPU Performance Evaluation[C]// International Conference on Learning Representations. 2015.

[14] FU H H, LIAO J, YANG J, et al. The Sunway Taihu Light supercomputer: system and applications[J]. Science China (Information Sciences), 2016, 7(59): 113-128.

[15] FANG J, FU H, ZHAO W, et al. swDNN: A Library for Accelerating Deep Learning Applications on Sunway TaihuLight [C]// 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 2017.

[16] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is All You Need[C]// Proceedings of the 31st International Conference on Neural Information Processing Systems. 2017, 30: 6000-6010.

[17] RADFORD A, NARASIMHAN K, SALIMANS T, et al. Improving language understanding by generative pre-training [EB/OL]. [2018-06-11]. https://openai.com/blog/language-unsupervised/.

[18] DEVLIN J, CHANG M, LEE K, et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [C]// North American Chapter of the Association for Computational Linguistics. 2018.

[19] RADFORD A, WU J, CHILD R, et al. Language models are unsupervised multitask learners[EB/OL]. [2019-02-14]. https://openai.com/blog/better-language-models/.

[20] MOHAMMAD S, MOSTOFA P, RAUL P, et al. Megatron-LM: Training Multi-Billion Parameter Language Models Using GPU Model Parallelism[J]. arXiv: 1909.08053, 2019.

[21] RAJBHANDARI S, RASLEY J, RUWASE O, et al. ZeRO: Memory Optimization Towards Training A Trillion Parameter Models[J]. arXiv: 1910.02054v2, 2020.

[22] BROWN T B, MANN B, RYDER N, et al. Language Models are Few-Shot Learners[J]. arXiv: 2005.14165, 2020.

[23] FEDUS W, ZOPH B, SHAZEER N, et al. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity[J]. arXiv: 2101.03961, 2021.

GAO Jie, born in 1997, postgraduate. His main research interests include high performance computing and deep learning.

LIU Xin, born in 1979, Ph.D, Ph.D supervisor, is a member of China Computer Federation. Her main research interests include parallel algorithms and parallel application software.

(责任编辑:柯颖)