中国科学技术大学

研究生学位论文中期进展报告

论文题目 基于国产AI处理器的Top-k算子的

设计与实现

学生姓名 张彪

学生学号 SA22218103

指导教师 薛美盛

实践导师（专业学位硕博生须填写） 吕秀全

所在院系 先进技术研究院

学科专业 电子信息

研究方向 计算机技术

填表日期 2024.10.30

中国科学技术大学研究生院培养办公室

二零二四年六月 制表

**说 明**

1. 抓好研究生学位论文中期考核工作是保证学位论文质量的一个重要环节。为加强对研究生培养的过程管理，规范研究生学位论文的中期进展报告，特印发此表。
2. 研究生需在学科点内报告，听取意见，进行论文中期考核。
3. 研究生论文中期考核通过后，在本表末签名后将此表复印件交所在学院教学办公室备查。

一．简况

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 研究生简况 | 姓 名 | | 张彪 | | | 学 号 | | SA22218103 | 出生年月 | 1999.03 |
| 性 别 | | 男 | | 身份证号 | 411424199903236219 | | | | |
| 入学时间 | | 2022.09 | | | | 培养方式 | | | ☑ 脱产 □ 不脱产 |
| 学生类别 | | | | * 学术硕士 ☑ 专业硕士 □ 学术博士 □ 专业博士 | | | | | |
| 论文内容和意义 | 题 目 | 中文 | | 基于国产AI处理器的Top-k算子的设计与实现 | | | | | | |
| 英文 | | Design and implementation of Top-k operator based on domestic AI processor | | | | | | |
| 摘 要 | 随着深度学习的不断发展，其模型的复杂度和规模呈指数级增长，传统CPU逐渐显露出其计算速度不足的弱点。为了应对这一挑战，各种AI处理器应运而生。Top-k算子用于从一组元素中选择排名在前k位置的元素，其在深度学习中有着广泛的应用。目前有多种算法实现Top-k算子，但是在不同的硬件体系结构上进行实现时，往往需要结合硬件自身的特点进行实现和优化才能发挥硬件的性能。因此，本课题计划基于目前国内外对于Top-k并行算法的研究，针对寒武纪自研AI处理器，完成Top-k算子的设计与实现。  在充分分析国产AI处理器体系结构和各Top-k算法特点的基础之上，选用合适的Top-k算法（radix Top-k和quick Top-k）在国产AI处理器上进行实现和优化；再对各Top-k算法在不同的数据类型和数据规模上的表现进行分析后，设计并实现Top-k算子；最后设计并完成Top-k算子在国产AI处理器上的测试模块。本课题主要工作内容和创新点总结如下：  1.调研Top-k并行算法并深入学习国产AI处理器体系结构和BangC编程。  2.基于AI处理器体系结构并行实现Top-k算法并优化。  3.设计并实现Top-k算子。 | | | | | | | | |
| 主题词 | 主题词数量不多于三个，主题词之间空一格（英文用“/ ”分隔） | | | | | | | | |
| 中文 | | AI处理器 Top-k算子 并行实现 | | | | | | |
| 英文 | | AI accelerator/Top-k operator/ Parallel implementation | | | | | | |

二．中期进展

|  |
| --- |
| １．阐述学位论文核心材料准备情况。  **1．论文提纲**  论文的正文部分暂定将分为五个章节来进行撰写，分别是绪论、相关技术介绍、基于国产AI处理器的Top-k算子设计实现、基于国产AI处理器的Top-k算子性能优化、基于国产AI处理器的Top-k 算子、测试验证总结与展望。  **第 1 章 绪论**  **1.1 研究背景及意义**  **1.2 Top-k 算法国内外研究现状**  **1.3 AI 处理器国内外发展现状**  **1.4 论文主要内容及章节安排**  **第 2 章 相关技术介绍**  **2.1 Top-k算法原理**  **2.2 PuDianNao处理器概述**  **2.3 本章小结**  **第 3 章 基于国产 AI 处理器的Top-k算子设计实现**  **3.1 Top-k算子计算流程**  **3.2 主机端数据准备**  **3.3 设备端Top-k算法实现**  **3.4 本章小结**  **第 4 章 基于国产AI处理器的Top-k算子性能优化**  **4.1 算子优化策略**  **4.2 Top-k 算子计算效率优化**  **4.3 Top-k 算子访存效率优化**  **4.4 本章小结**  **第 5 章 基于国产 AI 处理器的 Top-k 算子测试验证**  **5.1 实验环境与测试流程**  **5.2 Top-k 算子功能性能测试**  **5.3 Top-k 算子可用性测试**  **5.4 本章小结**  **第 6 章 总结与展望**  **6.1 本文工作总结**  **6.2 未来工作展望**  **参考文献**  **2．核心文献与本课题方向研究现状**  Top-k问题是一个经过充分研究的问题，已经提出了许多算法。针对该问题包括几个关键算法：基于优先队列的Top-k、基于排序的Top-k、bucket Top-k、radix Top-k以及bitonic Top-k。  优先队列方法被认为是一种解决Top-k问题的基本方法，其主要思想是在扫描输入向量的过程中，维护一个仅保留Top-k元素的优先队列。但是，在大规模并行处理器上实现这种方法时却比较局限，它需要对本地的优先队列进行多次维护，并且在合并各个本地优先级队列时，需要与全局内存进行全局同步，而且k的取值范围不能过大。  对于并行实现而言，另一种有效的替代方法是基于排序的方法，它通过对输入向量进行全排序后，然后从排序之后的向量中选择Top-k元素。但是这种方法可能会做一些多余的工作，比如对于不在1~k个元素范围内的元素也会进行排序，从而导致冗余计算。针对此问题，Alabi等人的研究指出，radix和bucket Top-k算法在某些情况下优于这种基于排序的解决方法，Bucket Top-k、radix Top-k和bitonic Top-k算法被以缓解排序选择中的效率问题。  其中，bucket Top-k首先获取输入向量的最小和最大值，然后将这个范围划分为多个桶，每个桶占据一个不相交的等值范围。通过扫描输入向量，将每个元素放入相应的桶中，并追踪每个桶中的元素数量，该方法可以轻松确定哪个桶包含第k个元素。该算法递归进行，直到递归的桶中只包含一个元素，即第k个元素。Radix Top-k类似于bucket Top-k，但它利用每个元素的bit数（基数）来确定值属于哪个桶。通过从最高有效位（MSD）到最低有效位（LSD）的迭代，算法只关注包含第k个元素的桶，从而减少不必要的计算。Bitonic Top-k则改进了传统的bitonic 排序算法，通过从大小为2k的双调序列中选择满足Top-k问题的k个元素，以减少工作负载。  **3．实验平台简介**  **3.1国产AI处理器硬件模型**  国产AI处理器是专为人工智能应用领域设计的专用处理器，通过精心设计的指令集、流水线、运算部件和访存部件，针对人工智能算法的计算特性和访存特性进行优化。其内部的MLU Core专注于人工智能中不同特征的访存数据流，通过设计专用的数据通路和运算部件，实现了对不同数据流之间的隔离。相较于通用处理器，MLU硬件在执行人工智能任务时展现出更高的性能、灵活性和能效比。同时，MLU硬件向软件提供了灵活的片上存储空间访问功能，以提高处理效率。  国产AI处理器硬件的基本组成单元是MLU Core，每个MLU Core都是一个具备完整计算、IO和控制功能的处理器核心。它既能够独立完成计算任务，也可以与其他MLU Core协同完成更大规模的计算任务。每4个MLU Core构成一个Cluster，在MLUv02及后续架构中，每个Cluster内还包含一个额外的Memory Core和一块被Memory Core和4个MLU Core共享的SRAM（Shared RAM，共享存储单元）。Memory Core无法执行向量和张量计算指令，其主要作用是在SRAM、DDR（Double Data Rate Synchronous Dynamic Random Access Memory，双倍速率同步动态随机存储器，通常简称为DDR）和MLU Core之间进行数据传输。整个抽象硬件模型划分为5个层级：服务器级、板卡级、芯片级、处理器簇（Cluster）级和MLU Core级。每个层次都包括抽象的控制单元、计算单元和存储单元，如图2.1所示，展示了Cambricon BANG异构计算平台的抽象硬件模型。第0级是服务器级，由多个CPU构成的控制单元、本地DDR存储单元和多个MLU板卡构成的计算单元组成。第1级是板卡级，每个MLU板卡由本地控制单元、DDR存储单元和MLU芯片构成的计算单元组成。第2级是芯片级，每个芯片由本地控制单元、本地存储单元（例如L2 Cache）以及存储单元（例如L2 Cache）以及一个或多个Cluster构成的计算单元组成。第3级    图3.1 [Cambricon BANG](https://www.cambricon.com/docs/sdk_1.13.0/cntoolkit_3.5.2/programming_guide_1.5.0/hardware_implementation/index.html" \l "hardwarehierarchy) 异构计算平台的抽象硬件模型  是Cluster级，每个Cluster由本地控制单元、共享存储以及多个MLU Core构成的计算单元组成。第4级是MLU Core级，每个MLU Core由本地控制单元、私有存储单元和计算单元组成。在MLU Core内部支持指令级并行和数据级并行。  抽象硬件模型提供了多层次的存储结构，如图2.2所示，包括GPR（General Purpose Register，通用寄存器）、NRAM、WRAM、SRAM、L2 Cache、LDRAM（Local DRAM，局部DRAM存储单元）、GDRAM（Global DRAM，全局DRAM存储空间）等。    图3.2抽象存储模型  **3.2国产AI处理器编程模型**  Cambricon BANG异构并行编程模型利用CPU和MLU协同计算，实现了 CPU 和 MLU 的优势互补。在Cambricon BANG异构并行编程模型中，CPU作为主机侧的控制设备，用于完成复杂的控制和任务调度；而设备侧的MLU则用于大规模并行计算和领域相关的计算任务。在BangC编程模型中，主机端的代码是串行执行的，设备端主要基于SIMD的并行形式执行。在MLU上执行的程序被称为核函数（Kernel）。每个在MLU core上执行的任务都会执行一次对应的Kernel 函数。在MLU上可以启动多个队列，这使得MLU可以同时执行多个并行的Kernel。如图2.3所示，主机侧代码顺序启动 Kernel1、Kernel2 和 Kernel3。    图3.3异构编程模型  具体执行顺序如下：  A.Kernel1和Kernel3位于同一个执行队列中，因此Kernel3需要等待Kernel1执行完毕才能开始执行。  B.Kernel2与Kernel1和Kernel3不在同一个队列中，因此Kernel2可以与Kernel1或Kernel3并行执行。 |
| ２．阐述学位论文核心内容推进情况。   1. **Top-k算子的设计与实现相关工作**   **1.1算子计算流程**  算法是一组计算步骤的抽象描述，算子则可以理解为计算工具，用于封装算法。不同于 CPU，AI 处理器不能作为单独的计算平台完成计算任务，而是需要和 CPU 协同处理计算任务。通常 CPU 作为主机端，负责处理串行任务和控制流程，而 AI 处理器作为设备端，负责完成大规模的并行计算任务。目前传统的 Top-k算法通常是基于CPU和高级编程语言设计实现的，得益于高级编程语言抽象程度高的特性，在实现基于CPU的Top-k算法时，编程者无需深入了解底层硬件的工作机制，因此可以大大降低编程的难度。同时许多高级编程语言拥有庞大的生态系统，这些生态系统提供了丰富的第三方库和框架，可以极大地提升编程的效率。然而随着数据规模的不断增大，使用传统的 CPU 进行数据处理已变得越来越乏力。 AI 处理器的硬件架构专为处理大规模数据设计，为充分优化 AI 处理器的通用编程能力，降低用户的编程难度，AI 处理器厂商会相应地推出适配于其产品的专用编程语言。不同于高级编程语言，适用于 AI 处理器的专用编程语言本质上更接近汇编语言。基于专用编程语言，可以在消耗更少资源的同时，实现许多非常精细的计算操作，从而高效地实现 Top-k 算法。但这也要求开发者在熟练掌握复杂底层代码语法的同时，对 AI 处理器架构细节具有较深的理解。  对于高级编程语言而言，其编译器和优化器已经内置了软件流水功能，可以自动地对编写的程序进行流水化优化，而使用专用编程语言要求开发者手动实现软件流水化。同时专用编程语言的语法规则和编程模式较为复杂，需要开发者具有较强的计算机原理和硬件知识，编写和调试过程相对困难。基于某国产 AI 芯片公司推出的 PuDianNao 架构AI处理器，并结合该公司推出的专用编程语言 BC，拟设计实现的 Top-k 算子计算流程方案如图 1.1所示。其中主机端负责完成输入参数检查和运行参数配置，然后将数据从主机端拷贝到设备端。设备端负责根据 Top-k 算法的原理，利用向量化计算指令，高效地完成 Top-k 算法的计算过程。    图 1.1算子计算流程图  **1.2 主机端设计与实现**  在 CPU 主机侧，主要涉及绑定句柄与队列和 MLU 设备、设置工作空间（workspace）、设置计算规模、检查参数内容。在完成这些操作后，即可通过CPU 端嵌入 MLU 设备侧，在 MLU 设备侧开始大规模的数据并行计算任务。   1. 接口设计及输入检查  |  |  |  | | --- | --- | --- | | 输入数据 | 数据类型 | 数据描述 | | input | int8/uint8/int16/int32/float16/float32 | 输入tensor，可以是任意维度 | | k | int32 | 表示取排行前k的效据，范围为[0,dim] | | dim | int32 | 表护对第dim维度进行操作，设input的维度总共有input\_dim维,则dim的取值范围为[-input\_dim, inputdim-1 ] | | largest | bool | 默认为true，控制取最大或者最小的值 | | sorted | bool | 默认为true的，控制是否返回排序好的值 | | lower\_index\_first | bool | 默认为false，控制排序是否稳定 |   表1.1输入参数表   |  |  |  | | --- | --- | --- | | 输出数据 | 数据类型 | 数据描述 | | output | int8/uint8/int16/int32/float16/float32 | 与input形状相比除了第dim维度长度为k其余与input形状相同，返同得到结果。 | | index | int16/int32  /uint16/uint32 | 与input形状相比除了第dim维度长度为k其余与input形状相同，表示的是对应input中的值在input中的位置。 |   表1.2输出参数表  2.任务类型选取  Top-K需要支持多维度数据可以表示如下:dim0，dime1，dim2，dim3。由左到右表示维度从高到低，按照dim可以将四个维度转化为:left，dim，right，left 为最高纬度，dim 为中间维度。当dim=3时表示对dim3 维度进行Top-K 查询,此时right = 1，left = dim0 \* dim1 \* dim2.当dim=0时,表示对dim0求 Top-K则对应的1ett 和 right 分别为:1，diml \* dim2 \* dim3。综上所述可以将数据处理分为left，dim，right，此时，针对right是否为1将分为两种情况。当right不等于1时，调用transpose算子将right和dim所在的维度进行转置，将待操作维度（dim）放在最低的维度，主要目的是在访问数据时保证数据是连续的，可以最大程度利用内存带宽。紧接着，最终的数据可以定义为：left\_right(length = left \* right)，dim两个维度。  MLU 硬件支持 Cluster 级、MLU Core 核心级、流水线级和数据级的并行计算。在利用MLU硬件进行计算之前，需计算出采用几个 Cluster 进行算子的任务计算，这些需要在 CPU 端配置任务规模和类型时根据数据规模，即left\_right，dim和k结合具体的算法实现方案来确定。  **1.3 设备端设计与实现**   1. quick Top-k kernel   算法原理：  function QuickSelect(arr, low, high, K):  pivot\_index = partition(arr, low, high)  If K < pivot\_index:  return QuickSelect(arr, low, pivot\_index - 1, K)  elif K > pivot\_index:  return QuickSelect(arr, pivot\_index + 1, high, K)  else:  return arr[pivot\_index]  # 分区过程  function partition(arr, low, high):  pivot = arr[high]  i = low - 1  for j from low to high - 1:  if arr[j] <= pivot:  i = i + 1  swap(arr[i], arr[j])  swap(arr[i + 1], arr[high])  return i + 1  2.任务类型  对于dim比较大的场景，需要多个core同时对一行数据进行处理，因此需要将任务类型设置为UnionX，考虑到top-k一般为多batch输入，且各batch之间数据没有关联，若是X值过大，将影响batch之间的并行度，并且有可能浪费某些板卡的算力。出于这两点考虑，将任务类型设为U1。    3.并行实现方案  1）每一个cluster每次处理一行数据，将数据均匀分到cluster上的每个core，每个core单次处理num个数据，需要repeat次将所分配的数据全部处理完。  2）单个core执行select函数，构建L\_local(小于privote的数)和R\_local(大于等于privote的数)。  3）执行cumsum（前缀和）和scatter(根据给定的索引和值更新数据)运算。获得L\_global，R\_global。  对于cumsum和scatter，有两种方案来进行实现：   1. 调用cumsum和scatter的kernel分别构建L\_global和R\_global。假设循环次数是 loop\_times次，则需要调用2 \* loop\_times次cumsum和2 \* loop\_times次scatetr分别求得L和R，因此共需要4 \* loop\_times针对显存的IO。 2. 将cumsum和scatter的实现逻辑内嵌到Top-k算子中，假设循环次数是 loop\_times次，仅需要2 \* loop\_times针对显存的IO。   2 .radix Top-k kernel实现  1.算法原理  function RadixTopK(data, K):  base = 10  buckets = [[] for \_ in range(base)]  max\_digit = get\_max\_digit(data)  top\_k\_data = []  for digit in [max\_digit-1 to 0]:  distribute\_to\_buckets(data, digit, buckets, base)  top\_k\_data\_t , data = merge\_buckets(buckets，k)  top\_k\_data.extend(top\_k\_data\_t)  If top\_k\_data.size + data.size == k:  Break;  buckets = [[] for \_ in range(base)]  top\_k\_data.extend(data)  return top\_k\_data  # 获取数据的最大位数  function get\_max\_digit(data):  max\_value = max(data)  return len(str(max\_value))  # 按当前位数分配到桶中  function distribute\_to\_buckets(data, digit, buckets, base):  for element in data:  current\_digit\_value = (element // (base \*\* digit)) % base  buckets[current\_digit\_value].append(element)  # 合并桶  function merge\_buckets(buckets):  merged\_data = []  for bucket in buckets:  If merged\_data.size + bucket.size >= K:  data = bucket  Break;  merged\_data.extend(bucket)  return merged\_data , data  2.任务类型  radix Top-k的并行实现方案需要多次启动核函数，并且数据规模随着核函数的启动次数而逐渐降低，因此该kernel的任务类型设置为UnionX，而X的具体数值根据当前数据规模的大小来进行决定。  3.并行实现方案  1）每一个job每次处理一行数据，将数据均匀分到job的每个core上，每个core单次处理num个数据，需要repeat次将所分配的数据全部处理完。  2）单个core执行select函数，构建L\_local(小于privote的数)和R\_local(大于等于privote的数)。  3）执行cumsum（前缀和）和scatter(根据给定的索引和值更新数据)运算。获得L\_global，R\_global。  对于cumsum和scatter，有两种方案来进行实现：   1. 调用cumsum和scatter的kernel分别构建L\_global和R\_global。假设循环次数是 loop\_times次，则需要调用2 \* loop\_times次cumsum和2 \* loop\_times次scatetr分别求得L和R，因此共需要4 \* loop\_times针对显存的IO。 2. 将cumsum和scatter的实现逻辑内嵌到Top-k算子中，假设循环次数是 loop\_times次，仅需要2 \* loop\_times针对显存的IO。     **2.Top-k算子的优化相关工作**  1.指令融合  MLU Core 内具有多条独立执行的指令流水线（以下简称为 PIPE），分别为负责标量运算的 ALU PIPE、负责向量运算的 VFU PIPE、负责张量运算的 TFU PIPE、负责片上和片外数据搬移的 IO-DMA PIPE 和负责片上数据搬移的 Move-DMA PIPE。如图 4.1所示，Inst-Cache 中的一段指令序列顺序开始执行后， 经过分发和调度被分配进了多个 PIPE 队列中，进入 PIPE 前指令是按顺序执行 译码的，而进入不同计算或访存队列后则由硬件解决寄存器重命名或读写依赖。 这意味着不同计算或者访存类型指令是可以并行执行的。BC 编程语言对部分计算功能提供了多种指令类型的接口，因此在 Top-k计算流程中可以通过选择合适的接口来实现指令流的并行。例如直方图计算步骤中用到 \_\_bang\_write\_value 指令在 Compute 流执行，同样可实现给指定内存段赋初值的 \_\_memset\_nram 指令则在 Move 流执行。通过使用\_\_memset\_nram 指令替代 \_\_bang\_write\_value，可以实现向量运算与内存段赋初值操作并行执行，减少总的指令运行耗时。同时 BC 编程语言还提供了多个指令融合操作接口，这类接口不仅可以减少数据中转和 NRAM 空间占用，还可以利用硬件的指令融合功能达到更高的性能。在整个 Top-k 算子的计算流程中，先后多次使用了 \_\_bang\_mul、\_\_bang\_add 和 \_\_bang\_sub 等向量操作指令。通过使用 \_\_bang\_fusion 指令融合接口，可以实现三个输入向量的乘加、乘减、加乘、减乘、减减、加加、加减、减加运算。从而进一步减少计算过程中用到的指令数量，提升算子的性能。   1. 软流水   实现计算与访存并行可以很好地掩盖计算或者访存的时间。前面提到 MLU Core 支持多指令流并行，在 Top-k算子的各计算阶段都涉及到加载输入数据、计算输入数据和存储计算结果。加载输入数据和存储计算结果时，指令被分配到数 据搬移队列; 计算输入数据时，指令则被分配到数据计算队列。因此对于有数据 依赖的计算以及访存指令，可以在使得它们在时序上分开，而对于无依赖的计算 以及访存指令，则可以同时并发执行，以减少指令等待时间，从而提高访存效率， 进而达到Top-k算子性能的目的。计算与访存并行通常都是通过排软件流水实现的。软件流水是一种特殊的指令重排，它将同一循环内不同循环迭代之间的 计算和访存进行重组。在重组后的循环中, 处理同一批数据的计算和访存被分散 到不同的循环迭代。软流水消除了同一循环迭代内的计算和访存之间的依赖关 系，使得计算和访存指令可以并行执行，从而相互隐藏执行时间，减少循环体的总执行时间。根据使用到的存储结构，软流水可分为三级流水和五级流水机制。   1. **Top-k算子的验证相关工作**   **4.1精度测试**  **基本原则：**相同的输入下，比较算子的Result（指国产AI处理器Top-k算子计算结果）和Baseline（指CPU或GPU上Top-k算子计算结果）  **主要步骤：**  A.使用Generator生成 Baseline (Generator 使用 Tensorflow、PyTorch 框架，调用框架对应算子计算 Baseline，Baseline 中包括输入数据、输出数据、对比方法)。  B.使用CNNL GTEST根据Baseline中的输入数据，调用CNNL算子API 计算Result。  当前比较Result和Baseline的方法拟采用2种：  A.静态阈值: 直接计算Result和Baseline的Diff（衡量差异的指标）值，使用绝对阈值判断 Result 是否正确 (不同类型算子的绝对阈值不同，Top-k拟采用0.003)  B.动态阈值: 使用框架用 fp64数据类型生成结果，设为Golden，然后分别计算 Result、Baseline与Golden 之间的 Diff 值，使用相对阈值判断 Result 是否正确 (不同类型算子的相对阈值不同，Top-k拟采用0.003)。  **两者的出发点差异：**  静态阈值: 直接计算Result和Baseline的 Diff 值 (Result和Baseline的数据类型相同)，这种方法假设 Tensorflow/PyTorch 的算子实现是正确的，且经过业界验证，相同输入下算子计算结果与 Tensorflow/PyTorch 算子计算结果的 Diff 值在阈值范围内，则认为算子实现正确。  image2022-3-29_15-51-13  动态阈值 : 使用框架用fp64生成 Golden，然后分别计算 Result、Baseline和 Golden之间的Diff值(Result和Baseline的数据类型相同, Golden恒为fp64)，这种方法假设 Tensorflow/PyTorch 的算子支持fp64，且fp64 的计算结果和真实值一致。相同输入下算子计算结果和真值的差距，与 Tensorflow/PyTorch 算子计算结果和真值的差距相当(Diff在一个数量级内)，则认为算子实现正确。  image2022-3-29_15-54-46  误差（Diff）计算公式：  平均相对误差：      均方相对误差：    在本节中，我们将对本文实现的Top-k算子和使用 cmath 库所得计算结果的精度进行比较分析，本文实现的非线性算子在多核深度学习处理器 DLP-M 上进行测试，cmath 库计算的结果则是在 CPU 环境下测试得出的。 测试环境的 CPU 配置为 112 个 Intel Xeon Gold 6330 处理器，每个处理器包 含 28 个核心，且具有 2.00 GHz 的主频，操作系统为 Ubuntu 20.04。由于Top-k算子仅涉及到具体的比较，因此精度需要与CPU下的运行结果保持完全一样。  **3.2性能测试**  我们将对比本文实现的Top-k算子、国产芯片的原有实现以及在NVIDIA A100 GPU 上实现的算子性能。其中，国产芯片上的原有实现方式主要根据冒泡算法，使用了国产芯片提供的硬件指令。本文实现的Top-k算子及国产芯片上的原有实现均在多核深度学习处理器 DLP-M 上进行了测试。对于 NVIDIA A100 GPU 上的Top-k算子，其实现主要依托于NVIDIA公司的cudnn和cublas算子库，选择在A100-PCIE-40GB 平台上进行性能测试。 |
|  |
|  |

三．已取得的与论文研究内容相关的成果

|  |
| --- |
| 已发表或已成文待发表的文章目录或其它相关研究成果。（**小论文全文请另附**） |

四．后续研究工作进度安排

|  |
| --- |
| **1．完成维护数据Cache一致性的操作**  当加速器向内存中存储数据完成之后可能造成有“脏数据”的隐患，因为加速器代理直接与内存进行数据交换，当交换的数据同时存在于数据Cache之中就会有这一隐患。目前已经有两个策略那就是加速器代理端同步维护一个目录，当发生这一问题时便可以立刻知晓，将数据Cache的指定标识位设置为脏位；第二个策略是不用耗费精力去维护目录，但是当加速器代理向内存中写入数据之后，利用龙芯内部指令“cacop/dbar”强制刷新指定内存数据，今后的工作是对于两种方法都进行实验操作来选中一个最适合的方法来维护数据Cache的一致性。  **2．进行更多元的实验设计**  应当考虑除量实验性能、待测结果准确率之外更加多元的实验设计，例如：当有多个加速器同时位于该套体系结构时，加速器代理是否会正确的进行加速器任务分配等等。同时应该再次进行性能优化，无论是Cpu与加速器代理的通信过程还是加速器代理与各个加速器的数据交互过程尽量做到低延迟。  **3．完成毕业论文的撰写** |

研究生本人签名： 图片包含 文本

描述已自动生成

2024年 10月 30日

中国科学技术大学研究生学位论文中期考核评审表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 姓 名 | 袁宇彤 | | 学 号 | | SA22218102 | 所在院系 | 先进技术研究院 |
| 学科、专业 | 电子信息 | | 研究方向 | | 计算机技术 | 指导教师 | 李诚（校内导师）  汪文祥（企业导师） |
| 拟撰写的学位论文题目 | | | 基于龙芯平台的加速器互联接口设计 | | | | |
| 学生类别 | | | * 学术硕士 ☑ 专业硕士 □ 学术博士 □ 专业博士 | | | | |
| 中期考核评审组成员名单 | | | | | | | |
| 姓名 | | 职称 | | 工作单位 | | | 签名 |
|  | |  | |  | | |  |
|  | |  | |  | | |  |
|  | |  | |  | | |  |
|  | |  | |  | | |  |
|  | |  | |  | | |  |
| 指导教师意见：  同意袁宇彤同学进行中期考核  图片包含 文本  描述已自动生成  指导教师签字： 实践导师签字（专业学位硕博生须签字）：  2024年 10月 30日 | | | | | | | |
| 评审小组意见：（是否通过中期考核，是否需要修改等）  评审小组组长签字：  年 月 日 | | | | | | | |

