专刊：先进智能计算平台及应用（下）
Special Issue: Advanced Intelligent Computing Platform and Application

ISSN 2096-742X
CN 10-1649/TP

文献CSTR:
32002.14.jfdc.
CN10-1649/
TP.2022.03.002

文献DOI:
10.11871/jfdc.issn.
2096-742X.2022.
03.002

文献PID:
21.86101.2/jfdc.
2096-742X.2022.
03.002

页码：19-29

获取全文

基于OpenCL的TensorFlow框架中Element-Wise算子实现

隋轶丞[1]，石昌青[1]，孙羽菲[1,2*]，张玉志[1,2]，陈禹乔[1]，张宇哲[1]

1. 南开大学，软件学院，天津 300350
2. 先进计算与关键软件海河实验室，天津 300350

摘　要：【目的】深度学习模型以较强的建模性能和优秀的多场景适应能力被广泛应用于各类典型人工智能领域。目前通常采用异构并行计算技术满足深度学习模型的算力需求，然而目前深度学习框架普遍使用CUDA或ROCm等编程模型，仅能支持特定厂商设备；对于通用异构计算设备，需要通过OpenCL编程标准实现支持，因此我们着力于实现TensorFlow框架的OpenCL版本。【方法】本文对TensorFlow框架中主要基于Eigen库提供的接口实现的Element-Wise算子进行代码分析，拆解对应结构体和类的封装方式，并基于OpenCL的编程标准对Element-Wise算子进行实现和封装，确保了代码的规范性和可扩展性。【结果】本文以CUDA算子为基准，对OpenCL的Element-Wise算子进行测试和对比，实验结果分别从正确性和计算效率两方面验证了本文OpenCL版本算子实现的可行性。【结论】作为实现OpenCL版本的TensorFlow框架这一工作的重要组成部分，本文成功实现了TensorFlow框架中Element-Wise算子的OpenCL版本，并经过实验验证了本文实现的算子的计算准确性和计算效率。

关键词：OpenCL; TensorFlow; 核函数

Implementation of Element-Wise Operator in TensorFlow Framework Based on OpenCL

SUI Yicheng[1], SHI Changqing[1], SUN Yufei[1,2*], ZHANG Yuzhi[1,2], CHEN Yuqiao[1], ZHANG Yuzhe[1]

1. College of software, Nankai University, Tianjin 300350, China
2. Haihe Lab of ITAI, Tianjin 300350, China

Abstract: **[Objective]** Deep learning models are widely used in various typical artificial intelligence fields with strong modeling performance and excellent multi-scene adaptability. At present, heterogeneous parallel computing technology is usually used to meet the computing power requirements of deep learning models. However, current deep learning frameworks generally use programming models such as CUDA or ROCm, which can only support equipment of specific manufacturers. It is necessary to support general heterogeneous computing equipment by the OpenCL programming standard. So we focused on implementing the OpenCL version of the TensorFlow framework. **[Methods]** This paper

基金项目：国家重点研发计划（2021YFB0300104）
*通信作者：孙羽菲（E-mail: yufei_sun@sina.com）

analyzes the code of the Element-Wise operator implemented mainly based on the interface provided by the Eigen library in the TensorFlow framework, disassembles the encapsulation method of the corresponding structure and class, and implements the Element-Wise operator based on the OpenCL programming standard. Implementation and encapsulation ensure the specification and extensibility of the code. **[Results]** This paper tests and compares the Element-Wise operator of OpenCL with the CUDA operator. The experimental results verify the feasibility of the implementation of the operator with OpenCL in terms of correctness and computational efficiency. **[Conclusions]** As an essential part of implementing the TensorFlow framework with OpenCL, the work presented in paper successfully implement the OpenCL version of the Element-Wise operator in the TensorFlow framework. The computational accuracy and calculation efficiency of the operator implemented in this paper is also verified by experiments.

Keywords: OpenCL; TensorFlow; Kernel Function

引　言

深度学习模型以较强的建模性能逐渐被广泛应用于各类典型人工智能应用场景中。近年来，随着深度学习的发展，模型的网络结构大幅增加，复杂的神经网络结构增加了深度学习模型的拟合效果，同时使得模型的训练成本与日俱增。目前通过 CPU 提供的算力性价比较低且总体算力不足，因此目前通常利用异构并行设备满足深度学习模型的算力需求。目前主流深度学习框架，如 TensorFlow[1]、PyTorch[2]、MXNet[3] 等，均实现了对 CUDA(Compute Unified Device Architecture)[4] 和 ROCm(Radeon Open Compute platform)[5] 编程模型的支持，利用 NVIDIA 和 AMD 厂商提供的强大 GPU 并行计算能力实现了对深度学习模型的加速。

然而国外厂商提供的 GPU 价格昂贵，在近年来的贸易禁运的背景之下，国内一些科研机构等被列入美国商务部的实体名单之中，主要基于国外厂商 GPU 设备进行加速的深度学习领域发展受到严重限制。基于如上原因，使用其他可控的异构计算设备加速深度学习计算具有重要意义。

目前主流深度学习框架还未实现可适配国产加速设备的编程模型的支持[6]。OpenCL(Open Computing Language)[7] 作为开放、统一的编程标准，被广泛应用于各类异构加速设备中，如 CPU、GPU、FPGA、DSP 等。基于 OpenCL，可以实现主流深度学习框架对不同厂商的多类型异构设备的支持。基于上述背景，TensorFlow 作为使用最为广泛的深度学习框架之一，实现其 OpenCL 版本具有极强的应用价值。

其中，Element-Wise 算子作为 TensorFlow 框架的重要组成部分，在 TensorFlow 框架中使用广泛，实现了较多的计算功能。然而不同于 TensorFlow 框架中常见的算子实现方式，Element-Wise 算子（包括一元算子和二元算子）通常通过调用 Eigen 库提供的接口完成算子计算功能的实现。其封装和实现方式与 TensorFlow 框架中常见的实现方式存在较大差别，在代码封装和结构上相对复杂，因此在保证可扩展性和代码规范性的前提下实现 OpenCL 版本的 Element-Wise 算子具有一定的实现难度。

本文针对 TensorFlow 框架中主要基于 Eigen 库实现的 Element-Wise 相关算子进行了分析，采用与 TensorFlow 源码中 Element-Wise 算子一致的模板类的形式进行了 OpenCL 版本的实现，并进行了实验验证。实验分别从计算正确性和计算性能两方面验证了本文转换方法的可行性。

1　背景与挑战

1.1　TensorFlow 框架

TensorFlow 是一个端到端的机器学习框架，被广泛应用于各类机器学习算法和深度神经网络的实

现中。TensorFlow 运行时包含 200 多个标准算子，包括数学计算、数组操作、控制流和状态管理等 [8]，其中许多算子是使用 Eigen::Tensor[9] 实现的，它使用 C++ 模板为多核 CPU 和 GPU 生成高效的并行代码。除此之外，其经常使用像 cuDNN[10] 这样的高性能并行计算库来实现特定算子，以更高效地实现某些特定类型运算。

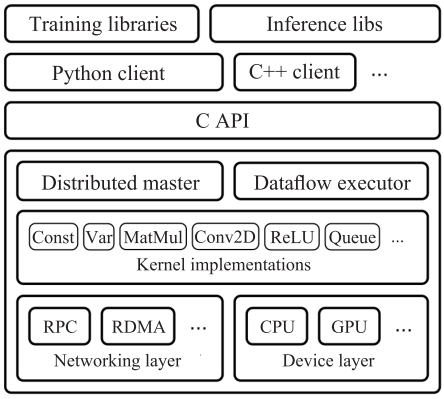

图 1 TensorFlow 架构图 [8]

Fig.1 TensorFlow architecture diagram[8]

TensorFlow 框架的架构如图 1 所示，从用户视角来看，其可以分为前端与后端：在前端部分，TensorFlow 框架提供基于 Python 或 C++ 语言的客户端，便于用户使用 TensorFlow 中各类网络结构实现构建复杂的神经网络模型；在后端部分，TensorFLow 将计算功能拆分为不同的算子，并分别基于 CPU、GPU 和 SYCL 编程标准等提供算子注册接口，并利用不同的核函数对不同编程模型版本的最底层提供代码实现。

TensorFlow 框架基于不同编程模型实现了对于一些异构设备的支持，其中主要支持的编程模型为 CUDA、ROCm 和 SYCL 编程标准。其中 SYCL 编程模型是一种免费的抽象编程模型，SYCL 使用与 CUDA 相似的单源编程模型使得设备端代码和主机端代码可以一起编译 [11]。SYCL 虽然允许基于 OpenCL 实现多种设备类型的支持，但需要底层 OpenCL 实现提供相应的扩展支持，因此无法基于 SYCL 广泛支持异构设备。除此之外，TensorFlow Lite 中也提供了对于 OpenCL 编程标准的支持，但 TensorFlow Lite 应用场景为移动端、嵌入式和物联网设备等，其作为一个轻量级框架仅能支持模型推理，无法利用不同厂商的加速设备基于 OpenCL 标准实现大型深度学习模型的训练。TensorFlow Lite 在设计和功能上都与 TensorFlow 框架存在较大差异，无法在 TensorFlow 框架中直接提供对于 OpenCL 编程标准的支持，与本文讨论的应用场景和目的都具有较大区别。

综上所述，需要实现 TensorFlow 框架的 OpenCL 版本以实现对不同异构设备的支持，实现利用不同异构并行设备加速深度学习模型的计算。

```
template <typename Device, typename Functor>
class UnaryOp : public OpKernel {
 public:
  typedef typename Functor::in_type Tin;
  typedef typename Functor::out_type Tout;
  explicit UnaryOp(OpKernelConstruction* ctx) : OpKernel(ctx) {
    auto in = DataTypeToEnum<Tin>::v();
    auto out = DataTypeToEnum<Tout>::v();
    OP_REQUIRES_OK(ctx, ctx->MatchSignature({in}, {out}));
  }
  void Compute(OpKernelContext* ctx) override {
    const Tensor& inp = ctx->input(0);
    Tensor* out = nullptr;
    if (std::is_same<Tin, Tout>::value) {
      OP_REQUIRES_OK(ctx, ctx->forward_input_or_allocate_output(
                              {0}, 0, inp.shape(), &out));
    } else {
      OP_REQUIRES_OK(ctx, ctx->allocate_output(0, inp.shape(),
&out));
    }
    functor::UnaryFunctor<Device, Functor>()(
        ctx->eigen_device<Device>(), out->flat<Tout>(),
inp.flat<Tin>());
  }
};
```

图 2 TensorFlow 中基于 Eigen 库的算子实现方式

Fig.2 Implementation of operators based on Eigen library in TensorFlow

1.2 Eigen 库介绍

Eigen 是一个高层次的 C ++ 库，有效支持线性代数、矩阵和矢量运算、数值分析及其相关的算法。Eigen 库对于 TensorFlow 框架而言，其一方面作为数据类型抽象，为 TensorFlow 的 Tensor 数据类型提

供了内存管理和底层映射；另一方面，Eigen 库中提供了对于 CPU、CUDA、ROCm 和 SYCL 编程标准的支持以及丰富的各类计算方法的实现接口，使得 TensorFlow 使用原生的 Tensor 数据类型通过简单数据类型转换即可调用 Eigen 库中的运算接口，极大地降低了各类运算的实现成本。

在 TensorFlow 的 Element-Wise 算子的注册过程中，通常使用 functor 命名空间下的各类结构体作为模板参数传入 UnaryOp 或 BinaryOp 中。如图 2 所示，TenosrFlow 中使用模板类实现基于 Eigen 接口调用的 UnaryOp 和 BinaryOp 两类算子。其中 Functor 作为模板参数，定义了具体运算的实现方法。

1.3 Element-Wise 算子实现的挑战

由于 TensorFlow 框架中的 Element-Wise 算子通常基于 Eigen 库提供的函数实现，与 TensorFlow 框架中通常基于 CUDA 核函数的实现方式存在较大差别。并且，由于 OpenCL 的运行时编译特性，其设备端代码通常需要以字符串的形式进行表示，因此需要对设备端代码设计封装方法以增加代码的规范性、减少代码量，因此实现 TensorFlow 框架中的 Element-Wise 算子具有如下挑战。

（1）运行时编译特性

OpenCL 由一门用于编写 kernels（在 OpenCL 设备上运行的函数）的语言（基于 C99）和一组用于定义并控制平台的 API 组成 [12]，其应用程序分为主机端代码和设备端代码。在 OpenCL1.2 版本中主机端代码使用通用编程语言（例如 C 或 C++）编写，并由传统编译器编译，在主机 CPU 上执行；设备端代码使用 C99 标准编写，存放在单独的 cl 文件中或编写成字符串，在主机端代码执行时动态加载编译成为二进制程序，然后与数据一起发送到 GPU 等加速设备上执行，执行结束后由主机端取回执行结果。因此，OpenCL 版本的算子实现过程中需要分别完成主机端代码和设备端代码，以实现内存拷贝、核函数编译和调用等过程。

除此之外 OpenCL 版本的算子实现过程还包括算子类的实现。在实现过程中，需要参照 CPU 版本和 GPU 版本的算子类实现代码，完成 OpenCL 版本算子类实现。并且由于数据类型的差异，需要将计算的数据处理为 OpenCL 支持的 cl_mem 的设备端内存类型，显式的数据类型处理进一步增加了 OpenCL 算子实现代码的复杂性。

（2）复杂的代码封装

在基于 Eigen 库接口所实现的 Element-Wise 相关算子中，由于结合了 Eigen 库中的相关接口，因此与常见的由“__global__”限定符标志的 CUDA 核函数代码不同，其通过 Eigen 库提供的接口直接实现封装计算功能。并且由于 Eigen 库是一种 C++ 模板库，不同于 TensorFlow 框架中常见的继承 OpKernel 类的实现方式，Element-Wise 算子中采用模板类的形式进行算子的实现与注册。

虽然通过将 Functor 作为模板参数传入的方式减少了代码量，增加了代码编写的便捷性，但对于 OpenCL 版本算子的实现则增加了工作量。在代码实现过程中，需要将上述模板参数中具体实现代码进行拆解，参照其各类计算功能的实现逻辑，采用 OpenCL 核函数的方式进行对应实现。并且由于 Element-Wise 算子种类较多，需要采用类似模板参数的方式完成对应代码的编写，以增加代码的可读性，便于后续版本的更新与扩展。

2 Element-Wise 算子的 OpenCL 实现

本节将介绍基于 OpenCL 编程标准实现 TensorFlow 中 Element-Wise 算子的方法。通过本文提出的实现方法，更多的支持 OpenCL 标准的异构计算设备能够支持 TensorFlow 框架，加速深度学习模型的计算。本文方法对于 PyTorch 和 MXNet 等主流深度学习框架中 Element-Wise 相关算子的 OpenCL

版本实现也具有参考意义。

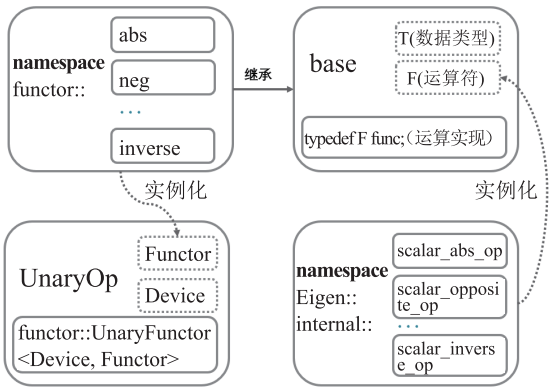

图 3 Element–Wise 算子实现示意图

Fig.3 Schematic diagram of Element-Wise operator implementation

2.1 Element-Wise 算子实现方式

Element-Wise 算子利用模板参数实现了不同计算功能，为了保证算子实现的规范性并减少代码量、确保可扩展性，我们分析了源码中的实现方式，并基于 OpenCL 实现了具体运算功能的核函数字符串。除此之外，基于 OpenCL 运行时编译的特性，我们在实现了代码封装的同时，对于多次调用的设备端代码也采用函数封装的方式实现设备端代码字符串的生成，进一步简化了代码的复杂度。本节将分别从源码中 Functor 结构体实现、OpenCL 版本的 Functor 结构体实现和运算符实现方式展示 OpenCL 版本 Element-Wise 算子的实现方式。

（1）Functor 结构体的继承与实例化

如图 2 所示，Element-Wise 相关算子通过模板参数 Functor 实例化 UnaryOp 类中的 UnaryFunctor 结构体实现对应算子类，通过 Functor 结构体中包含的设备端代码实现对应的运算符功能。

上述结构体和类的关系如图 3 所示，其中 UnaryOp 类中的模板参数 Functor 在命名空间 functor 下定义，其通过继承 base 类型模板结构体的方式，使用 func 对应具体运算的实现。base 类型作为模板结构体，通过使用命名空间 Eigen::internal 下的 Eigen 库中的运算符或在 TensorFlow 源码中继承实现的原生运算符，使得 Functor 模板参数对应的类可以获取对应算子的具体实现代码。

（2）Functor 结构体的 OpenCL 实现

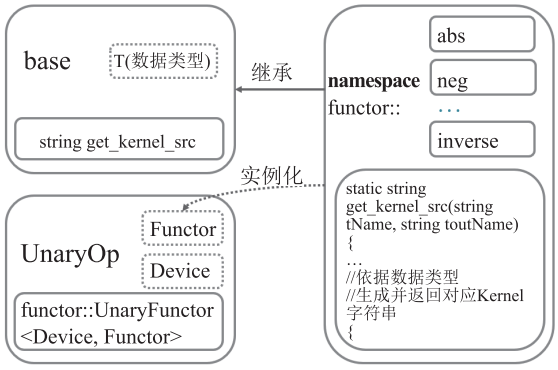

图 4 Element–Wise 算子 OpenCL 实现示意图

Fig.4 Schematic diagram of Element-Wise operator implementation based on OpenCL

参照如图 3 所示源码中 Element-Wise 算子实现方式，本文通过提取与 CUDA 代码无关的封装功能，并对 OpenCL 实现中涉及的核函数运算部分进行封装，通过相似的继承方式实现模板类的实例化。

通过 Functor 模板参数，我们利用命名空间 functor 下的结构体实例化对应的 OpenCL 版本的 UnaryOp 或 BinaryOp 类，进而实现 OpenCL 版本的 Element-Wise 算子。具体来说，如图 4 所示，我们在分析 OpenCL 执行特点后，重写了 base 结构体，利用函数 get_kernel_src 在不同的子类中实现对应的 OpenCL Kernel 字符串生成功能，其依据传入的输入数据类型和输出数据类型，生成完成对应功能、使用不同数据类型的 OpenCL 字符串。

（3）OpenCL 运算符实现方式

除此之外，由于 CUDA、ROCm 和 SYCL 编程标准均为单源编程模型，在使用 Eigen::internal 命名空间下的相关算子实现的结构体时，其重载的“()”运算符中通常采用调用其他算子的结构体中的函数辅助完成对应运算。以调用相对简单的“zeta”运算为例，在实现过程中，调用了 abs、machep 等函数。对此，我们将对应的多次出现在多个核函数中的常用功能函数封装为一个单独函数，以生成对应功能的 OpenCL 设备端代码字符串。

上述封装方式在保证功能实现的同时，增强了代码的规范性，抽象出共同的相似功能，减少了代码量。同时，基于模板结构体实现的方式也保证了算子实现的可扩展性，即对于未来 TensorFlow 新版本中出现的新算子，也可以采用类似的方式进行实现并扩展填充。

2.2 实现示例

```
template <typename T>
struct scalar_logistic_op {
  EIGEN_EMPTY_STRUCT_CTOR(scalar_logistic_op)
  EIGEN_DEVICE_FUNC EIGEN_STRONG_INLINE T operator()(const T&
x) const {
    const T one = T(1);
    return one / (one + numext::exp(-x));
  }

  template <typename Packet> EIGEN_DEVICE_FUNC
EIGEN_STRONG_INLINE
  Packet packetOp(const Packet& x) const {
    const Packet one = pset1<Packet>(T(1));
    return pdiv(one, padd(one, pexp(pnegate(x))));
  }
};
```

图 5 Eigen 相关运算符实现

Fig.5 Implementation of operators related to Eigen

本节以 sigmoid 算子为例，展示了 Element-Wise 算子的 OpenCL 版本实现方法。源码中的实现如图 5 所示，其通过采用 scalar_logistic_op 结构体实现了具体的 sigmoid 算子的计算功能。

如图 6 所示，我们基于 OpenCL 编程标准，通过字符串的方式实现了 sigmoid 运算的 OpenCL Kernel，并按照前文介绍的封装方法，将其作为结构体中的成员函数进行封装，实现利用传入的不同数据类型，生成对应的 Kernel 函数字符串。从而在 OpenCL 的设备端代码中实现类似 C++ 中模板参数的功能。

在算子执行过程中，TensorFlow 可以通过模板参数 Functor 获得对应的结构体 scalar_logistic_op，并利用其中的静态函数 get_kernel_src 获取 OpenCL 设备端代码字符串，基于 OpenCL 提供的主机端 API 将字符串编译成 OpenCL 程序并获取 OpenCL 核函数以执行计算，最终实现 sigmoid 算子的计算功能。

```
template <typename T>
struct sin : base<T> {
static string get_kernel_src(string tName, string toutName) {
string src1 = " __kernel void Kernel(__global const "+tName+" *
input, __global "+ toutName +" * output, int N) \
{\
const float cutoff_upper = 15.6437711715698242f; \
 ... }";
string src2 = " __kernel void Kernel(__global const "+tName+" *
input, __global "+ toutName +" * output, int N)\
{\
const float cutoff_upper = 36.736800569677104f;
...\}";
if(tName == "_Complex float" || toutName == "_Complex double")
    return src2;
else
    return src1;
}
};
```

图 6 OpenCL 实现示例

Fig.6 Implementation example based on OpenCL

3 实验验证

表 1 实验环境配置

Table 1 Experimental environment settings

设备类型	软硬件类型	参数
NVIDIA GPU 服务器	CPU	Intel Xeon Gold 5218 CPU @ 2.30GHz
	RAM	192GB
	GPU	NVIDIA Tesla V100S
	NVIDIA CUDA Toolkit	10.2
	OpenCL	NVIDIA CUDA: OpenCL 1.2

3.1 实验环境

由于 NVIDIA GPU 设备同时支持 CUDA 编程模型和 OpenCL 编程标准，可以提供基于相同设备的不同编程模型，便于测试本文提出方法的计算效率与准确性。因此本文基于 NVIDIA GPU 设备完成实验。

本文使用的实验环境具体如表 1 所示，本文使用 CUDA 的 OpenCL 实现进行实验验证，并与 TensorFlow 框架中基于 CUDA 实现的 Element-Wise 算子进行对比。

3.2 计算正确性验证

本文基于上述方法共实现了 81 个 Element-Wise 算子，其中包括 BinaryOp 类型算子 35 个，UnaryOp 类型算子 46 个，本节对所有已实现算子进行了正确性分析，分别在不同的误差精度和数据类型下进行了实验，涵盖了对应算子常用的基本数据类型。本文以 CUDA 版本 TensorFlow 中 raw_ops[13] 标准接口的 Element-Wise 算子计算结果为基准，对比本文实现的 OpenCL 版本算子计算结果，使用 unittest[14] 进行对比测试。

对于整数类型，根据两者的计算结果是否完全一致判断是否通过；对于浮点类型，由于不同的编译器和硬件对精度的支持情况不同，可能会导致两者的计算结果存在精度差异，对此本文设定了三种相对误差范围，在误差范围内判断是否通过。正确性实验将 81 个算子分别在不同类型和不同误差精度下进行测试，测试通过率计算公式如下所示，实验结果如表 2、表 3 所示，通过率公式为：

$$通过率 = \frac{测试算子总数 - 测试失败算子数量}{测试算子总数} * 100\% \quad (1)$$

表 2 整型数据准确性测试

Table 2 Accuracy test of integer data

类型	Int16	Int32	Int64
通过率	100%	100%	100%

表 3 浮点型数据准确性测试

Table 3 Accuracy test of floating point data

类型/精度	1e-3	1e-4	1e-5
Float32	100%	100%	100%
Float64	100%	100%	100%

实验结果证明，本文提出的方法是正确可靠的，所有已实现算子均能够通过测试或在指定误差范围内通过测试，能够达到标准的误差精度，确保深度学习模型计算过程中的正确性。

3.3 计算速度比较

在算子的性能分析中，本文选取了较为典型的 8 个算子，包括 UnaryOp 和 BinaryOp 两类情形，具体算子名称及其功能如表 4 所示。

表 4 算子介绍

Table 4 Operator introduction

算子名称	功能
Atan	求解张量的三角函数值Atan
Conj	求解复数的共轭值
Invert	在二进制位上进行反转
LogicalNot	求解张量的逻辑非
BitwiseXor	计算两张量的xor
Equal	判断两这个能量是否相等
RealDiv	计算实数类型的二进制x/y值
AddV2	计算两张量二进制位相加

分别对如上 8 个算子进行了性能测试，根据算子的注册情况，测试数据类型选用了深度学习框架中常用的 float32 数据类型，测试数据量从 2 的 10 次方至 2 的 18 次方逐渐递增，覆盖了深度学习模型中常见的数据量。本文通过对比基于 OpenCL 实现的算子和 TensorFlow 中基于 CUDA 实现的算子在不同数据量下的计算用时，分析算子的计算性能差异，上述 8 个算子在不同数据量下的计算用时如图 7 所示。

从实验结果可以看出，随着测试数据量的递增，TensorFlow 中基于 CUDA 和 Eigen 库实现的 Element-Wise 算子表现较为稳定，本文基于 OpenCL 实现的 Element-Wise 算子耗时在数据量规模较小时用时无明显增长，在数据量规模较大时计算用时线性增长。

3.4 实验结果分析

本文提出的方法首次实现了 TensorFlow 框架中 Element-Wise 算子的 OpenCL 版本，为 TensorFlow 框架的 OpenCL 版本实现提供了坚实基础，确保了

Element-Wise 的运算在更多异构计算设备上的应用。实验结果验证了基于 OpenCL 实现的 Element-Wise 算子的计算正确性。

从计算性能上来看，这些算子的时间复杂度虽然都为 O(n)，但本文实现的 OpenCL 版本算子与 CUDA 相比仍有一定的性能差距。其中 OpenCL 版本的算子在不同量级的实验数据下耗时高于 CUDA 版本的算子，在数据量较小时与 CUDA 性能相当，当数据量过大时与 CUDA 相比性能差距较大。

OpenCL 作为一种开放、通用的编程标准，在通用性和代码移植性上具有较强的优势，而各厂商提供的 OpenCL 实现中由于考虑通用性标准，无法实现针对各自设备体系结构特点的优化。CUDA 由 NVIDIA 设计，根据 NVIDIA 设备体系结构特点实现了深入的优化。CUDA 在线程数更大时针对硬件计算单元分配和调度提供了较好的优化，因此 CUDA

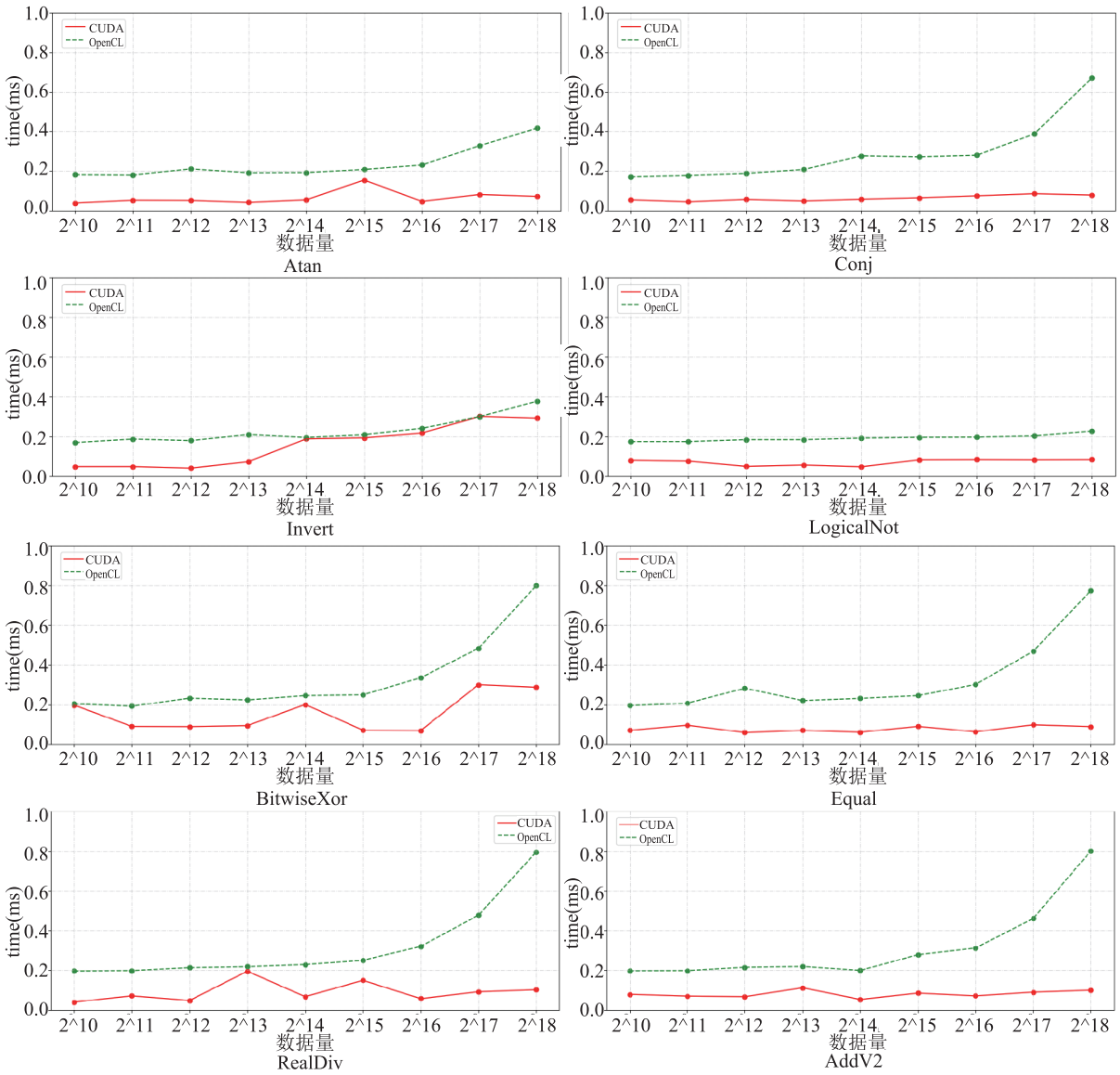

图 7 计算性能对比

Fig.7 Comparison of computational performance

在处理更大数据量时计算用时没有明显增加。为了使得 OpenCL 实现与 CUDA 具有相当的性能，仍有很多技术难点需要进一步研究和攻克。未来我们将继续探索和优化深度学习框架中 OpenCL 版本算子的实现方案，优化 OpenCL 算子的核函数编写与调用过程，提高基于 OpenCL 标准的设备端代码计算效率。

4 结论与展望

本文针对 TensorFlow 框架中 Element-Wise 算子中的 OpenCL 版本实现方式进行了讨论，分析了 Eigen 库支持的三类设备 CPU、GPU 和 SYCL 的编程方式，讨论了基于 OpenCL 实现 TensorFlow 对多种异构设备支持的必要性。分析了 TensorFlow 中 Element-Wise 相关算子通过 Eigen 库实现的方式，并介绍了基于相似封装方式的 OpenCL 版本的算子和运算符实现和封装方法。作为实现 OpenCL 版本的 TensorFlow 框架这一工作的重要组成部分，本文成功实现了 TensorFlow 框架中 Element-Wise 算子的 OpenCL 版本。

本文的实验部分测试了 TensorFlow 中 Element-Wise 相关算子的 OpenCL 版本与 CUDA 版本的计算结果和计算用时，并比较计算正确性和计算用时。实验证明，我们通过 OpenCL 标准实现的相关算子准确性合格。在常用的数据范围下，OpenCL 版本与 CUDA 版本的实现性能相当，在数据量较大时，OpenCL 版本与 CUDA 版本在性能上存在一定差距。

在后续的工作中，我们将针对具体硬件设备探索优化方法，提升 OpenCL 版本 Element-Wise 算子的性能，并将本文方法扩展至 PyTorch、MXNet 等其他主流深度学习框架的 Element-Wise 算子的实现中。

利益冲突声明

所有作者声明不存在利益冲突关系。

参考文献

[1] Abadi M, Barham P, Chen J, et al. Tensorflow: A system for large-scale machine learning[C]//12th USENIX symposium on operating systems design and implement-ation ({OSDI} 16), 2016:265-283.

[2] Paszke A, Gross S, Massa F, et al. PyTorch: an imperative style, high-performance deep learning library[C]//Proceedings of the 33rd International Conference on Neural Information Processing Systems, 2019: 8026-8037.

[3] Chen T, Li M, Li Y, et al. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems [J]. arXiv preprint arXiv:1512.0274, 2015.

[4] Cheng J. CUDA by Example: An Introduction to General-Purpose GPU Programming[J]. Scalable Computing: Practice and Experience, 2010, 11(4): 401-401.

[5] AMD ROCm™ Documents[EB/OL].[2022-02-17]. https://rocmdocs.amd.com/en/latest/index.html.

[6] 丁立德.支持国产计算平台的深度学习加速技术研究[D].中国电子科技集团公司电子科学研究院,2020. DOI: 10.27728/d.cnki.gdzkx.2020.000010.

[7] Munshi A. The opencl specification[C]//2009 IEEE Hot Chips 21 Symposium (HCS), IEEE, 2009: 1-314.

[8] Jung K H. A study on machine learning for steganalysis [C]//Proceedings of the 3rd International Conference on Machine Learning and Soft Computing, 2019: 12-15.

[9] Guennebaud G, Jacob B. Eigen[EB/OL]. [2022-02-15]. http://eigen. tuxfamily. org.

[10] Chetlur S, Woolley C, Vandermersch P, et al. cudnn: Efficient primitives for deep learning[J]. arXiv preprint

arXiv:1410.0759, 2014.

[11] Goli M, Iwanski L, Richards A. Accelerated machine learning using TensorFlow and SYCL on OpenCL Devices [C]//Proceedings of the 5th International Work-shop on OpenCL,2017: 1-4.

[12] Fang J, Huang C, Tang T, et al. Parallel programming models for heterogeneous many-cores: a comprehensive survey[J]. CCF Transactions on High Performance Computing, 2020, 2(4): 382-400.

[13] Python Software Foundation. unittest introduction [EB/OL]. [2022-02-17].https://docs.python.org/zh-cn/3.7/library/unittest.html.

[14] Tensorflow. Public API for tf.raw_ops namespace. [EB/OL]. [2022-02-17].https://www.tensorflow.org/api_docs/python/tf/raw_ops.

收稿日期：2022 年 3 月 4 日

隋轶丞，南开大学，软件学院，博士研究生，主要研究方向为人工智能。
本文中负责方法设计、代码实现与论文撰写。

SUI Yicheng is a Ph.D. student in College of Software at Nankai Univer-sity. His research interests include artificial intelligence.
In this article, he is responsible for method design, code implementation, and paper writing.

E-mail: suiyicheng@mail.nankai.edu.cn

石昌青，南开大学，软件学院，硕士研究生，主要研究方向为深度学习与高性能计算。
本文中他参与方法设计与承担实验测试。

SHI Changqing is a master's student in College of Software at Nankai University. His research interests include deep learning and high-performance computing.
In this paper, he participated in method design and undertook experimental tests.

E-mail: shichangqing@mail.nankai.edu.cn

孙羽菲，南开大学，软件学院，特聘研究员，博士，先进计算与关键软件海河实验室研究员，主要研究方向为深度学习、异构计算、人工智能等。
本文主要承担论文指导和修改工作。

SUN Yufei, Ph.D., is a professor at College of Software, Nankai University and a professor at HaiHe Lab of ITAI. Her research interests include deep learning, heterogeneous computing, artificial intelligence, etc.
In this paper, she is mainly responsible for the paper guidance and paper revision.

E-mail: yufei_sun@sina.com

张玉志，南开大学，讲席教授，软件学院，院长，先进计算与关键软件海河实验室顶尖人才，主要研究方向为人工智能、模式识别、自然语言处理等。
本文主要承担文献调研及指导。

ZHANG Yuzhi is the chair professor and the Dean of Software College in Nankai University. He is the top-level Expert in HaiHe Lab of ITAI. His research interests focus on artificial intelligence, pattern recognition, natural language processing, etc.
In this paper, he is mainly responsible for the related work investigation and guidance.

E-mail: zyz@nankai.edu.cn

陈禹乔，南开大学，软件学院，硕士研究生，主要研究方向为深度学习框架移植与高性能计算。

本文主要参与方案设计与实验。

CHEN Yuqiao is currently a master's student in the College of Software at Nankai University, Tianjin, China. His research interests include deep learning framework transplantation and high-performance computing.

In this paper, he participated in scheme design and experiment.

E-mail: ujoenk@mail.nankai.edu.cn

张宇哲，南开大学，软件学院，硕士研究生，主要研究方向为人工智能。

本文主要参与方案设计与实验。

ZHANG Yuzhe is currently a master's student in the College of Software at Nankai University, Tianjin, China. His research interests include artificial intelligence.

In this paper, he participated in scheme design and experiment.

E-mail: zyzcs@mail.nankai.edu.cn

隋轶丞,石昌青,孙羽菲,张玉志,陈禹乔,张宇哲.基于OpenCL的TensorFlow框架中Element-Wise算子实现[J]. 数据与计算发展前沿, 2022,4(3):19-29. CSTR: 32002.14.jfdc.CN10-1649/TP.2022.03.002.DOI:10.11871/jfdc.issn.2096-742X.2022.03.002.PID:21.86101.2/jfdc.2096-742X.2022.03.002.

SUI Yicheng, SHI Changqing, SUN Yufei, ZHANG Yuzhi, CHEN Yuqiao, ZHANG Yuzhe. Implementation of Element-Wise Operator in TensorFlow Framework Based on OpenCL [J]. *Frontiers of Data & Computing*, 2022,4(3):19-29. CSTR: 32002.14.jfdc.CN10-1649/TP.2022.03.002.DOI:10.11871/jfdc.issn.2096-742X.2022.03.002.PID:21.86101.2/jfdc.2096-742X.2022.03.002.