文章编号:1671-6833(2023)01-0031-07

基于国产 PuDianNao 芯片的向量函数库优化

杨指政[1], 杜子东[2], 文渊博[3]

(1.郑州大学 河南先进技术研究院,河南 郑州 450001;2.中国科学院计算技术研究所 计算机体系结构国家重点实验室,北京 100190;3.中国科学技术大学 计算机学院,安徽 合肥 230026)

摘 要:目前国产人工智能处理器 PuDianNao 芯片上的向量数学函数只能依靠循环调用标量函数来实现,该方法性能比较低。基于 PuDianNao 芯片提出了 3 种优化方法。方法一为插值方法;方法二为 SIMD 加掩码方法;方法三基于 PuDianNao 的硬件阵列结构,使用 VLIW 指令操作阵列中的每个处理单元,封装出 SIMT 编程模型,提出了暴露分支范围和分支扁平化的编程方法。对以上 3 种方法进行精度和性能测试,对比实验结果表明,方法三具有最好的精度和性能。使用方法三实现基于国产 PuDianNao 芯片的向量数学函数库 PuDianNao-VecMath,解决了数学函数多分支结构难以向量化的难题。该函数库精度性能较好、功能稳定、运行正确,提供的接口包括取整函数、超越函数、比较函数、激活函数等常见基础数学库函数。在精度上,将函数定义域区间全数据作为输入,运算结果和标量函数在 CPU i7 运行的结果进行对比。结果表明,单精度版本最大 ULP 值为 2,半精度版本最大 ULP 值为 1。性能与使用标量循环相比有较大提高,单精度版本相对于标量循环平均加速比平均值为 18.26,最大加速比为 35.90;半精度版本平均加速比平均值为 15.65,最大加速比为 30.11。

关键词:向量化函数; PuDianNao-VecMath; 国产人工智能处理器; 暴露分支范围和分支扁平化

中图分类号:TP311 **文献标志码**:A **doi**:10.13705/j.issn.1671-6833.2023.01.013

基础数学库函数是计算机系统中最为基础和重要的函数库,生活中各种需要对数据进行处理计算的场景都会用到基础数学函数库。在航天、航空电子等诸多领域中的嵌入式软件通常涉及数学库函数(如 sin( )、tanh( )等)来实现复杂的计算。大到人工智能、智慧城市、智慧医疗,小到日常工作生活,基础数学库函数都提供了重要的数据计算支撑。随着数学库函数计算数据量的增大,提高向量数学函数计算的性能越来越重要,越来越多的向量数学函数得到了广泛的应用。比如英特尔 MKL(math kernel library)库中的 VM 提供了非常丰富的数学向量化函数[1]。Intel Short Vector Math Library[2]是商业界有名的函数库,该库提供了高度优化的子程序,用于评估基本函数,这些子程序可以使用 Intel 处理器中可用的多种向量扩展,但是这种库是专有的,仅针对 Intel 处理器进行了优化。AMD 提供了一个向量化的 libm,称为 AMD 核心数据库[3]。Anand 等[4]实现了 32 个单精度函数来调整 Cell BE SPU 计算引擎,使用了名为 Coconut 的环境,该环境支持模式的快速原型设计、汇编语言片段和模式的快速单元测试。Lauter[5]发布了一个用纯 C 实现的开源 Vectorlibm 库。VDT 数学库[6]是为编译器的自动矢量化功能编写的数学库。实现这些向量化函数库大多采用两种方法:①人为使用汇编指令拼接或者封装一些底层的向量指令;②用编写高级语言程序使用编译器提供的自动向量化来实现[7]。本文主要工作如下。

(1)使用插值方法实现向量数学函数,向量中的每个元素只需要 1 次乘法运算和 1 次加法运算即可模拟出函数运算结果,但由于使用分段直线模拟曲线的方式,精度较差。

(2)使用 SIMD(single instruction multiple data)

收稿日期:2022-02-24;**修订日期**:2022-06-27

基金项目:国家自然科学基金资助项目(61925208);国家自然科学基金联合基金资助项目(U19B2019);中国科学院战略性先导科技专项(XDB32050200);北京智源人工智能研究院以及北京市科技新星计划项目(Z191100001119093)

通信作者:杜子东(1989—),男,河南南阳人,中国科学院计算技术研究所副研究员,博士,主要从事计算机体系结构的研究,E-mail:duzidong@ict.ac.cn。

引用本文:杨指政, 杜子东, 文渊博. 基于国产 PuDianNao 芯片的向量函数库优化[J]. 郑州大学学报(工学版),2023,44(1):31-37. (YANG Z Z, DU Z D, WEN Y B. Optimization of vector function library based on domestic PuDianNao chip[J]. Journal of Zhengzhou university (engineering science),2023,44(1):31-37.)

加掩码编程方法对向量函数进行优化,解决了数学函数中多分支结构无法向量化的问题,但是使用该方法时大部分复杂数学函数中有临时变量需要申请临时空间,临时空间在向量化函数中变为临时向量空间,所需空间成倍增长,会占用智能处理器宝贵的计算空间。

(3)基于 PuDianNao 芯片封装出 SIMT 编程模型,提出了暴露分支范围和分支扁平化的编程方法(方法三),解决了数学函数多分支结构难以向量化的问题,充分发挥国产 PuDianNao 芯片性能优势。对比实验表明,该方法具有较高的性能和精度,因此本文采用此方法实现向量数学函数库。

(4)基于方法三实现了国产深度学习处理器 PuDianNao 芯片向量数学库 PuDianNao-VecMath。该函数库精度性能较好、功能稳定、运行正确。

1 研究背景

1.1 国产 PuDianNao 处理器

PuDianNao 是一种 ML(machine learning)加速器,包含 7 种代表性的 ML 技术,包括 k-means、k-最近邻、朴素贝叶斯、支持向量机、线性回归、分类树和深度神经网络。PuDianNao 通过对不同 ML 技术的计算原语和局部性属性进行深入分析,可以在 3.51 mm^2 的面积上执行高达 1 056 GOP/s 的运算(如加法和乘法),仅消耗能量 596 mW。与 NVIDIA K20M GPU(28 nm 工艺)相比,PuDianNao(65 nm 工艺)的速度为其速度的 1.20 倍,能耗为其能耗的 1/128.41。

如图 1 所示,PuDianNao 由若干个功能单元(FUs)、3 个数据缓冲区(HotBuf、ColdBuf 和 OutputBuf)、1 个指令缓冲区(InstBuf)、1 个控制模块和 1 个 DMA 组成。功能单元(FU)是 PuDianNao 的基本执行单元,每个 FU 由两部分组成:1 个机器学习功能单元 MLU(maching learning unit)和 1 个算术逻辑单元 ALU(arithmetic logic unit)。相比之前针对小范围机器学习技术设计的机器学习加速器,PuDianNao 在数据特征变化或应用场景发生变化时更加健壮,因为其为用户提供了一系列候选技术。

1.2 PuDianNao 编程模型

1.2.1 VLIW 指令集架构

PuDianNao 编程模型中的一种指令集使用 VLIW(very long instruction word)。VLIW 中的每条指令完成多个独立操作,将多个相互没有依赖的指令封装到一条超长的指令字中并且使相应数量的 ALU 完成指令的一系列操作,使用编译器来控制指令的调度并且解决指令之间的依赖性。

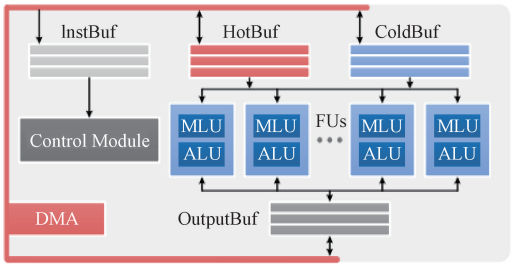

图 1 PuDianNao 的加速器架构[8]

Figure 1 Accelerator architecture of PuDianNao[8]

VLIW 在一条指令中封装多个并行操作选项,使用 VLIW 指令操作阵列结构中的处理单元。指令的长度比常见的 RISC(reduced instruction set compute)和 CISC(complex instruction set computer)指令长度长,因此叫做超长指令集。典型的商用 VLIW 芯片架构有 Intel IA-64,是 Itanium 系列 64 位微处理器指令体系结构[9];ATI GPU 采用 VLIW 架构,每个 TP 是一个 5-way 的 VLIW Processor[10];移动应用领域 DSP 采用 VLIW 架构并且取得了成功[11]。

1.2.2 SIMT 编程模型

SIMT 指的是单指令多线程,有别于 SISD(单指令单数据)、SIMD(单指令多数据),SIMT 的 1 条指令可以处理多条线程和多条数据,图 2 展示了 SISD、SIMD 和 SIMT 的区别[12]。由于 PuDianNao 硬件向量单元本身可以控制每个小核是否执行任务,在此基础上实现了简单的掩码翻译器,给 PuDianNao 的硬件单元增加了 SIMT 指令,并且增加 SIMT 指令的译码器,用 VLIW 指令集封装出 SIMT 编程模型。使用 SIMT 编程模型编程时,只需要描述 1 个线程的行为,所有线程以锁步的方式并行执行同样的指令。这种方式能批量处理数据,只需要写 1 份指令,硬件会同时运行 1 份相同的指令,实现计算的并行化。

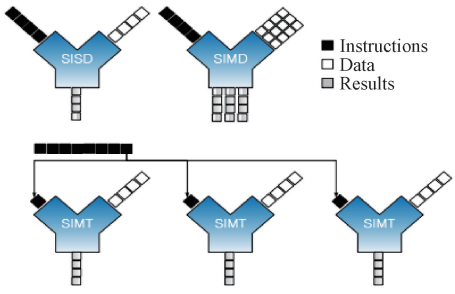

图 2 SISD、SIMD、SIMT 的区别[12]

Figure 2 Difference among SISD, SIMD, and SIMT[12]

图 3 为 PuDianNao 封装出的 SIMT 编程模型示意图,其结构是一种阵列结构,包括 M 行 N 列的处理单元,每个处理单元中包括子运算单元和子存储单元,其中子运算单元可以完成逻辑运算、算术运算

等,子存储单元可用于存储数据和谓词信息等。

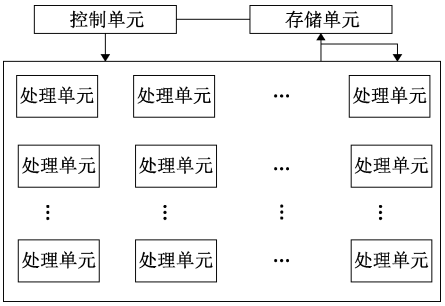

图 3　PuDianNao 中的 SIMT 编程模型

Figure 3　SIMT programming model in PuDianNao

每个处理单元可以执行 1 个线程。处理单元中的子存储单元的多个谓词寄存器决定每个分支是否使能。每个处理单元处理不同的输入数据,所执行的分支可能不同,处理单元的行为也可能不同,以此来实现多线程。阵列结构保证了并行性,阵列结构大小为 $M\times N$,每次读取 $M\times N$ 个元素,计算 $M\times N$ 个数据。子存储单元里面的谓词寄存器用于分支限定,子存储单元里面的存储寄存器存储中间运算结果和临时变量;子运算单元进行运算。使用 VLIW 指令来控制每个处理单元的各种操作,比如输入、输出、数据交互以及计算。

1.2.3　SIMD 指令集

PuDianNao 处理器支持 SIMD 指令集。SIMD 是单指令多数据流,用 1 个控制器控制多个处理单元,时间上的并行性是通过对数据中的每个元素进行相同的计算来完成的。在执行 SIMD 指令时,通过 1 条指令可以同时对多个元素进行计算。比如 1 个 128 bit 的向量寄存器,每次编译的时候可以对 4 个 float 类型的数据进行计算。

PuDianNao 的 SIMD 指令集支持非常丰富的指令,不同的指令相互拼接能够完成许多复杂的功能。支持的指令包括配置指令、算术运算指令、比较指令、逻辑和移位指令、谓词指令、扩展精度的算数运算指令、数据移动指令等。

2　实现方法

2.1　插值方法

激活函数(如 tanh、relu、sigmoid 等)使用分段直线模拟曲线求解。将函数分成多个小段,每段插入多个点使用最小二乘法拟合,每段使用直线模拟曲线,当所分的每一段足够小的时候,误差可以非常小。将斜率和截距存储在激活表中,分段区间存储在常数表中,省去了函数中逻辑判断的部分,通过查表的方式来确定输入区间,对于每个分段使用硬件查表单元选择合适的斜率 k 和截距 b,通过 1 次乘法和加法运算可以得到函数运算结果,结果较好地拟合原函数。

2.2　SIMD 加掩码方法

使用 SIMD 指令直接对标量函数进行向量化面临以下两个主要的挑战。

(1)条件分支判断。在标量程序中多个条件分支可以使用 if、else 等语句写出来,然而在向量函数中,每个元素所走的分支可能不同,不能用 SIMD 的方式让所有元素执行同一个流程。

(2)临时变量。在标量程序中临时变量可以存储在寄存器中。但是在向量程序中,临时变量变为临时向量,占用的空间较大,并且不能存储在寄存器中,必须存储在智能计算器空间中,所需的临时空间成倍增长,占用宝贵的计算资源。

针对条件分支判断问题采用 SIMD 加掩码的方法解决:使用浮点数 0.0 和 1.0 作为掩码,使用 SIMD 指令将所有的分支都遍历一遍,每个分支所得结果乘以掩码将无效数据清零,最后将各分支有效结果相加拼接出向量输出。

对于临时变量问题,部分简单函数临时向量可以复用输出向量的空间,无须申请临时空间;然而大部分复杂函数无法复用输出向量空间,需要大量的临时空间。

2.3　SIMT 编程模型方法

基于 PuDianNao 芯片上封装出的 SIMT 编程模型,提出暴露分支范围和分支扁平化来解决数学函数多分支结构难以向量化的难题。注释标注出标量函数中各个复杂嵌套分支的范围,使用谓词寄存器来存储条件分支。SIMT 编程模型中的指令会把所有分支都遍历一遍,使用谓词寄存器控制各分支的语句是否执行,若该分支执行,在该分支上使用对应的汇编指令,最终在 ALU 计算单元对元素进行计算和处理。

异构编程模型通常由通用处理器和多个特定于域的处理器组成[13]:通用处理器叫做主机端(host),用于复杂的控制和调度,主要的任务包括设备获取、数据或者参数准备、执行流创建、任务描述、内核启动和输出获取等;特定于域的处理器作为子设备,用于大规模的并行计算和特定于域的计算任务,二者合作共同完成计算任务。由于 device 端(智能处理器)不能像 host 端(通用处理器)那样在特定的分支上返回结果,每个分支的结果都会暂时存储在寄存器里,最终返回寄存器中的值。

数学函数中大部分复杂函数具有多分支、分支

嵌套的特性。SIMT 编程模型不支持跳转指令,需要把函数的所有分支遍历一遍。这时,SIMT 编程变得困难并且容易出现错误。针对这种问题提出了暴露分支范围和分支扁平化的方法。暴露分支范围就是由于标量函数中使用的嵌套的 if、else 语句块往往难以将每个分支的具体范围分辨清楚,可以人为地将每个分支的具体上限和下限作为注释标注出来。分支扁平化的具体方法是使用存储寄存器存储变量和中间计算结果,使用通过扁平后的分支构造谓词寄存器中的分支条件,或者使用谓词寄存器中已经存在条件的逻辑运算(与或非等)的结合来构造复杂条件分支,并且在相应的条件分支下编写汇编指令执行标量函数中的对应操作。

图 4 为标量函数中的多分支结构使用暴露分支范围和分支扁平化的方法转化为 SIMT 编程模型中对应谓词寄存器的映射过程。最左边为高级语言中的多分支条件;中间为暴露分支范围的结果,p 代表每个分支条件,p 之后为每个分支的具体上下限;右边为分支扁平化过程,将暴露的分支范围用汇编指令表示出来。

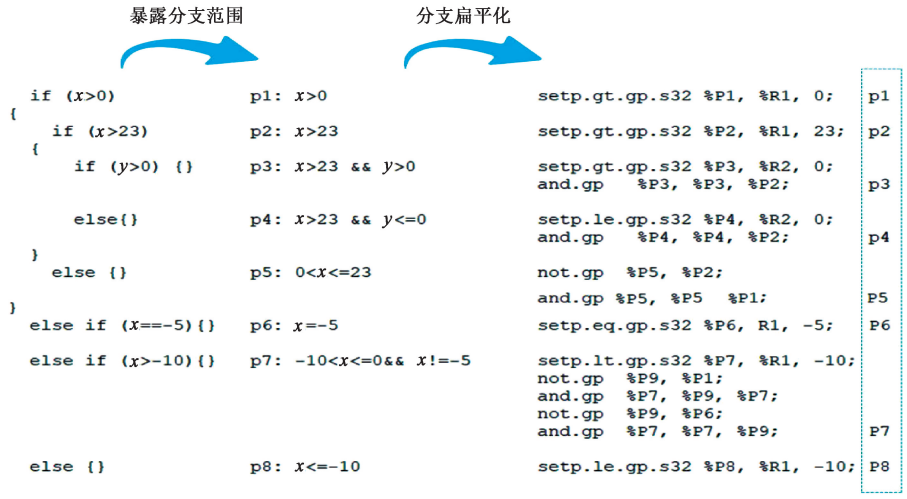

图 4 暴露分支范围和分支扁平化过程

Figure 4 Expose branch scope and branch flattening process

标量函数中的多分支结构使用暴露分支范围和分支扁平化的方法转化为 SIMT 编程模型中对应谓词寄存器中的条件,使用 SIMT 编程模型中阵列结构的每个处理单元操作单个数据,根据处理单元中的多个谓词寄存器决定每个分支是否执行,同时并行处理多个输入数据,每个处理单元相当于运行一个线程,实现计算并行化。

3 优劣分析实验

3.1 实验介绍

本文实验硬件平台基于国产 PuDianNao 处理器、CPU i7 和 GPU Tesla T4。向量函数汇编代码采用 PuDianNao 芯片自研指令集进行编写。编译工具链等软件使用 PuDianNao 芯片配套软件栈。

(1)精度统计方法。使用输入数据类型所能表示的所有数据和函数定义域的交集作为输入,测试最大 ULP(unit in last place)[14]。这种方法的缺点是测试时间较长,比如单精度浮点数据用 32 位数据表示,全数据区域一共包含 2^{32} 个数据;优点是测试的数据完整,最大 ULP 比较准确。

(2)求 ULP。使用 ULP 衡量函数库误差。挑选部分函数在 PuDianNao 芯片上运行,得到的运行结果和 glibc 库[15]函数在 CPU i7 中运行结果进行对比,利用 ULP 计算公式进行计算。ULP 表示浮点数之间的距离,值越大代表误差越大。对于特殊值 NAN,INF 的处理方式和 glibc 库函数保持一致。

(3)性能测试。函数定义域全数据区域取不同规模输入数据,测试出计算完所有输入数据所用的时间作为性能数据。使用加速比(原性能时间/新性能时间)来衡量性能加速情况。

3.2 插值方法分析

如图 5 所示,选取 6 个复杂函数,分别测试在 SIMT 编程模型、CUDA MATH API[16]、标量循环、SIMD 加掩码方法、插值方法的最大 ULP 值。结果表明,插值方法最大 ULP 值为 7,最小 ULP 值为 3,使用插值法测试的 ULP 值最大,精度最差。

表 1 为测试出的 7 个函数中 SIMT 编程模型相对于插值方法和 SIMD 方法的平均加速比和最大加速比。测试结果表明,SIMT 编程模型相对于插值方法的平均加速比平均值为 4.13,最大加速比为 6.38,

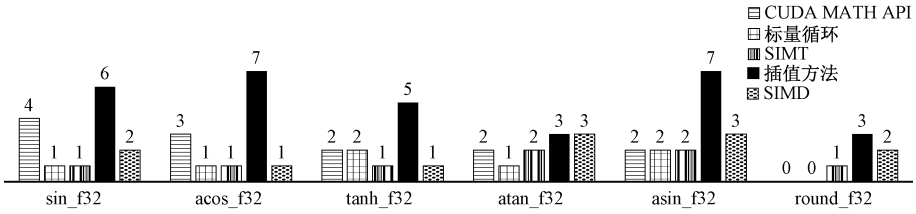

图 5　各种方法的 ULP 值对比

Figure 5　Comparison of ULP values of various methods

表 1　SIMT 编程模型相对于 SIMD 加掩码方法和插值方法的加速比

Table 1　Speedup of SIMT programming model relative to SIMD masking method and interpolation method

函数	SIMT 相对于 SIMD		SIMT 相对于插值方法	
	平均加速比	最大加速比	平均加速比	最大加速比
sin_f32	1.25	2.11	4.25	5.68
asin_f32	2.47	3.32	5.11	5.81
tanh_f32	1.72	2.14	5.21	6.24
atan_f32	1.65	1.85	1.89	2.65
acos_f32	1.30	1.86	3.21	3.64
round_f32	1.60	2.14	4.02	4.68
max_f32	1.52	1.91	5.24	6.38

插值方法性能不如 SIMT 编程模型。

插值方法由于使用常数表存储分段区间,激活表存储每个分段的斜率和截距,省去了函数逻辑判断的部分,通过查表的方式来确定输入区间,只需要一次乘法和加法可以得到函数模拟运算结果,但使用直线模拟曲线,始终存在误差,精度较差。

3.3　SIMD 加掩码优化分析

表 1 结果表明,SIMT 方法相对于 SIMD 方法的平均加速比平均值为 1.64,最大加速比为 3.32,SIMD 加掩码方法性能不如 SIMT 编程模型。

图 5 中的结果表明 SIMD 加掩码方法最大 ULP 值为 3,最小 ULP 值为 1,ULP 值在测试函数中均大于等于 SIMT 编程模型方法,精度不如 SIMT 编程模型方法。

SIMD 加掩码方法需要占用大量的临时空间,以 asin 函数为例,限定短向量长度为 64 位,所需要的临时空间为 6 784 字节,若短向量长度增加,所需要的临时空间会同比例增加。

该方法性能精度不如 SIMT 编程模型方法,并且大部分函数需要申请临时空间,在向量函数中临时空间成倍增长,并且需要占用智能处理器的空间,浪费大量宝贵的计算资源。

3.4　SIMT 编程模型优化分析

精度对比实验使用 glibc 库[15] 函数在 CPU i7 上运行的结果和 PuDianNao-VecMath 库中对应函数进行误差计算,得出最大 ULP 值,并和 CUDA MATH API 函数库公开的 ULP 值进行比较。

性能对比实验分为两组:①使用循环调用标量函数性能作为基准,测试 PuDianNao-VecMath 库中多个函数相对于基准的性能加速比;②使用 CUDA MATH API 函数库中选定的函数在 GPU Tesla T4 上的运行性能作为基准,测试出 PuDianNao-VecMath 库中对应函数在 PuDianNao 芯片上相对于基准的加速比。

3.4.1　向量函数精度测试

图 5 结果表明,PuDianNao-VecMath 函数库中测试函数的最大 ULP 值大部分小于等于 CUDA MATH API 库中的函数,某些函数的 ULP 值小于标量循环。PuDianNao-VecMath 库中单精度浮点数(float)最大 ULP 为 2,具有较好的精度。

由于半精度浮点数(half)所表示数的指数位和小数位相对于单精度浮点数(float)均较短,为了提高计算精度、减小误差,结合 PuDianNao 芯片的硬件特性,采用单精度计算来实现。实验结果表明,半精度浮点数版本函数最大 ULP 不超过 1。

3.4.2　向量函数性能测试

(1)使用循环调用标量函数作为性能基准,测试 PuDianNao-VecMath 库中多个函数相对于基准方法的加速比。

表 2 表明,PuDianNao-VecMath 库中的单精度浮点数函数相对于标量循环方法均有较高的加速比,所测几个函数平均加速比平均值为 18.26,最大加速比为 35.90;PuDianNao-VecMath 库中的半精度浮点数函数相对于标量循环方法均有较高的加速比,所测函数的平均加速比平均值为 15.65,最大加速比为 30.11。

表 2　PuDianNao-VecMath 库相对于标量循环的加速比

Table 2　Speedup of PuDianNao-VecMath library relative to scalar loop

函数	平均加速比	最大加速比	数学函数	平均加速比	最大加速比
sin_f32	18.62	20.10	sin_f16	15.42	18.22
asin_f32	33.50	35.90	asin_f16	28.50	30.11
tanh_f32	11.33	12.30	tanh_f16	9.60	10.80
atan_f32	17.43	18.30	atan_f16	15.65	16.54
acos_f32	7.16	7.80	acos_f16	5.36	7.11
round_f32	20.13	20.70	round_f16	18.25	19.82
max_f32	19.67	20.20	max_f16	16.80	17.31

(2)使用 CUDA MATH API 函数库函数在 GPU Tesla T4 上运行时间作为性能基准,测 PuDianNao-VecMath 库在 PuDianNao 处理器上相对于基准方法的加速比。

在大数据段,输入浮点数的个数超过 1 M 时,选取 7 个复杂函数测试,结果如表 3 所示。可以看出,PuDianNao-VecMath 在 PuDianNao 硬件上运行,相对于 CUDA MATH API 中对应测试函数在 GPU Tesla T4 硬件上的平均加速比平均值为 1.64,最大加速比为 3.32。

表 3 PuDianNao-VecMath 库相对于 CUDAMATH API 的加速比

Table 3 Speedup of PuDianNao-VecMath library relative to CUDA MATH API

函数	平均加速比	最大加速比
sin_f32	1.25	2.11
asin_f32	2.47	3.32
tanh_f32	1.72	2.14
atan_f32	1.65	1.85
acos_f32	1.30	1.86
round_f32	1.60	2.14
max_f32	1.52	1.91

SIMT 编程模型性能和精度较好,不需要额外的临时空间,使用 SIMT 编程模型下的专用的存储寄存器和谓词寄存器存储中间变量和条件分支。因此本文的 PuDianNao-VecMath 函数库使用 SIMT 编程模型的方法来实现。

综上可得出 SIMT 编程模型具有最好的性能和精度,且不占用智能处理器空间。可以作为 PuDianNao 芯片上向量数学函数库 PuDianNao-VecMath 的实现方法。

4 结论

本文基于国产 PuDianNao 处理器提出了 3 种向量化函数优化方法,经过精度和性能测试分析,得出各种优化方法的优劣。

(1)插值方法。所测函数中 SIMT 方法相对于插值法的平均加速比平均值为 4.13,最大加速比为 6.38,性能一般;插值方法最大 ULP 值为 7,最小 ULP 值为 3,精度差、实现方法简单。该方法适用于对精度要求不高、实现方法要求简单的场景。

(2) SIMD 加掩码方法。SIMT 方法相对于 SIMD 方法的平均加速比平均值为 1.64,最大加速比为 3.32;SIMD 最大 ULP 值为 3,最小 ULP 值为 1。该方法性能和精度均较好,但需要大量的智能处理器临时空间,实现方法较为复杂,适用于精度和性能要求较高、对智能处理器空间限制不高、具有较为全面的 SIMD 指令支持的场景。

(3)SIMT 编程模型方法。性能和精度均最好,但实现较为复杂,且需要有较为全面的汇编指令支持,适用于性能和精度要求高的场景。对 PuDianNao-VecMath 函数库中选定一些实现较为复杂的函数进行精度和性能测试。单精度版本函数最大 ULP 不超过 2,半精度版本函数最大 ULP 不超过 1。选定该库中某些较为复杂函数和标量循环对比,单精度浮点数平均加速比平均值为 18.26,最大加速比为 35.90;半精度版本平均加速比平均值为 15.65,最大加速比为 30.11。单精度版本和 GPU Tesla T4 对比平均加速比平均值为 1.64,最大加速比为 3.32。

测试结果表明,使用 SIMT 编程模型方法构建的 PuDianNao-VecMath 函数库性能精度较好、功能稳定、运行正确,充分发挥了国产 PuDianNao 智能处理器的性能优势,为 PuDianNao 处理器提供了向量数学函数库支持,泛化了人工智能处理器的应用场景。

参考文献:

[1] Intel Corporation. Intel® oneAPI math kernel library [EB/OL]. (2019-11-01)[2022-02-10]. https://software.intel.com/en-us/mkl.

[2] Intel Corporation. Intel short vector math library [EB/OL]. (2021-06-20)[2022-02-10]. https://software.intel.com/en-us/node/523613.

[3] Advanced Micro Devices, Inc.. AMD core math library [EB/OL]. (2013-07-24)[2022-02-10]. http://developer.amd.com/tools-and-sdks/archive/acml-product-features/.

[4] ANAND C K, KAHL W. An optimized cell BE special function library generated by coconut[J]. IEEE transactions on computers, 2009, 58(8): 1126-1138.

[5] LAUTER C. A new open-source SIMD vector libm fully implemented with high-level scalar C[C]//2016 50th Asilomar Conference on Signals, Systems and Computers. Piscataway: IEEE, 2016: 407-411.

[6] PIPARO D, INNOCENTE V, HAUTH T. Speeding up HEP experiment software with a library of fast and auto-vectorisable mathematical functions[J]. Journal of physics: conference series, 2014, 513(5): 052027.

[7] 刘聃, 郭绍忠, 郝江伟, 等. 基于 SIMD 扩展部件的长向量超越函数实现方法[J]. 计算机科学, 2021, 48(6): 26-33.
LIU D, GUO S Z, HAO J W, et al. Implementation of transcendental functions on vectors based on SIMD exten-

sions[J]. Computer science, 2021, 48(6): 26-33.
[8] LIU D F, CHEN T S, LIU S L, et al. PuDianNao[J]. ACM SIGPLAN notices, 2015, 50(4): 369-381.
[9] HUCK J, MORRIS D, ROSS J, et al. Introducing the IA-64 architecture[J]. IEEE micro, 2000, 20(5): 12-23.
[10] ZHANG Y, HU Y, LI B, et al. Performance and power analysis of ATI GPU: a statistical approach[C]//2011 IEEE Sixth International Conference on Networking, Architecture, and Storage. Piscataway: IEEE, 2011: 149-158.
[11] KUMURA T, IKEKAWA M, YOSBIDA M, et al. VLIW DSP for mobile applications[J]. IEEE signal processing magazine, 2002, 19(4): 10-21.
[12] KYUNG G, JUNG C M, LEE K. An implementation of a SIMT architecture-based stream processor[C]//TENCON 2014-2014 IEEE Region 10 Conference. Piscataway: IEEE, 2014: 1-5.
[13] XIONG Y Q. A unified programming model for heterogeneous computing with CPU and accelerator technologies[C]//12th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI). Piscataway: IEEE, 2019: 1-4.
[14] MULLER J K. On the definition of ulp(x)[R/OL]. (2005-02-01)[2022-02-10]. https://www.researchgate.net/publication/236944278_On_the_definition_of_ulpx.
[15] Free Software Foundation. The GNU C Library (glibc)[EB/OL]. (2019-11-01)[2022-02-10]. https://www.gnu.org/software/libc/.
[16] NVIDIA. CUDA Math API[EB/OL]. (2022-01-12)[2022-02-10]. https://docs.nvidia.com/cuda/cuda-math-api/index.html.

Optimization of Vector Function Library Based on Domestic PuDianNao Chip

YANG Zhizheng[1], DU Zidong[2], WEN Yuanbo[3]

(1. Henan Institute of Advanced Technology, Zhengzhou University, Zhengzhou 450001; 2. State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190; 3. School of Computer, University of Science and Technology of China, Hefei 230026)

Abstract: At present, the vector math functions on the PuDianNao chip of the domestic artificial intelligence processor can only be implemented by calling scalar functions cyclically, and the performance of this method is relatively low. Based on the PuDianNao chip, three optimization methods were proposed. The first two were interpolation method and SIMD masking method. Thirdly, based on a hardware array structure on PuDianNao, VLIW instructions were used to operate each processing unit in the array, and the SIMT programming model was encapsulated programmatically. The accuracy and performance of the above three methods were tested, and the experimental results showed that the third method had the best accuracy and performance. The third method was used to implement the vector mathematical function library PuDianNao-VecMath based on the domestic PuDianNao chip, which solved the problem that the multi-branch structure of mathematical functions was difficult to vectorize. The function library had good precision performance, stable functions and correct operation. The provided interfaces included rounding functions, transcendental functions, comparison functions, activation functions and other common basic math library functions. In terms of precision, the entire data of the function definition domain interval was used as input, and the operation result was compared with the result of the scalar function running on the CPU i7. The results showed that the maximum ULP value was 2, and the maximum ULP value of the half-precision version was 1. Compared with the use of scalar loop, the performance was greatly improved. Compared with the scalar loop, the single-precision version had an average speed-up ratio of 18.26 and a maximum speed-up ratio of 35.90. The half-precision version had an average speed-up ratio of 15.65 and a maximum speed-up ratio of 30.11.

Keywords: vectorized function; PuDianNao-VecMath; domestic artificial intelligence processor; expose branch scope and branch flattening