doi:10.3772/j.issn.1002-0470.2024.04.006

基于紧耦合加速器的高性能 Java 压缩系统①

王 雪②*** 李文青*** 张婷婷*** 张福新*** 王 剑③*** 敖 琪***

(*处理器芯片全国重点实验室(中国科学院计算技术研究所) 北京 100190)

(**中国科学院大学 北京 100049)

(***龙芯中科技术有限公司 北京 100190)

摘 要 Java 无损压缩应用广泛,尽管软件算法在不断改进,但仍然存在压缩速度慢、耗时严重等问题。本文使用领域前沿的紧耦合框架集成压缩加速器的处理器平台,设计了一个高性能 Java 压缩系统,在 Java 虚拟机(JVM)内部实现了对紧耦合无损压缩加速器的封装,并为其提供了轻量级运行时环境。本系统可以有效减少通信开销、避免数据拷贝问题,具有易编程、快速压缩的特点,充分发挥加速器给 Java 压缩带来的性能优势。实验结果表明,此系统大幅提升了 Java 压缩性能,压缩速度达到主流 Java Gzip 软件压缩的 63 倍,最高可达 247 倍,且在大数据集下性能提升更显著。

关键词 Java 压缩; 无损压缩; 紧耦合加速器; Java 虚拟机(JVM)

Java 是最流行的编程语言之一,因其具有简单、安全、面向对象和平台无关等特点,被广泛用来开发各种服务器和终端的软件。随着大数据时代的到来,数据量爆炸式增长,数据压缩在数据高效的存储和传输中起着关键作用[1-2]。因此,让 Java 程序员能够便捷地使用高性能压缩功能是当下需要解决的重要问题。

Java 虚拟机(Java virtual machine,JVM)和 Java 类库是决定 Java 软件性能和开发效率的重要因素[3]。JVM 为 Java 提供了底层运行时环境,是 Java 程序可以跨平台高效执行的基础。Java 类库提供了丰富的应用程序编程接口(application programming interface,API),其功能完善,极大地提高了 Java 程序员的开发效率。无损压缩是 Java 开发的核心功能之一,为了让 Java 程序员便捷地使用无损压缩功能,Java 标准类库和第三方类库提供了多种压缩 API[4-5],实现了各种常用的无损压缩算法。软件开发人员可以根据实际业务需求,选择合适的压缩算法。

尽管无损压缩算法在 Java 类库中已有很好的支持,并且算法在不断改进,但是纯软件压缩依然非常耗时,越来越多的人开始关注硬件压缩。随着专用硬件加速器的流行,近几年涌现出许多压缩加速器[6-13]。遗憾的是,压缩加速器大多采用松耦合的集成方式,中央处理器(central processing unit,CPU)和加速器之间通信开销较大[14],使得 JVM 使用松耦合压缩加速器的性能收益有限。目前,Java 程序员使用压缩加速器并充分发挥其性能优势主要面临如下挑战:(1)如何使压缩加速器能够在提供高性能压缩功能的同时,避免过高的通信开销;(2)如何在异构系统中降低 Java 程序员使用硬件加速器的编程难度。

为了解决以上问题,本文使用领域前沿的紧耦合加速器设计框架 TCADer[15]集成压缩加速器的处理器平台[16],设计了一个高性能 Java 压缩系统。本系统将 Java 应用程序调用压缩加速器的过程进行

① 国家重点研发计划(2022YFB3105103)资助项目。

② 女,1993 年生,博士生;研究方向:计算机系统结构,Java 虚拟机;E-mail: wangxue19b@ict.ac.cn。

③ 通信作者,E-mail: jw@ict.ac.cn。

(收稿日期:2023-04-02)

了透明封装,并为其提供了轻量级运行时环境。实验结果表明,此系统能够显著提高压缩性能,并且使用简单,不会增加程序员的编程难度。

论文剩余部分组织如下。第 1 节总结了相关工作;第 2 节介绍 Java 压缩和 TCADer 框架集成紧耦合压缩加速器的处理器平台;第 3 节描述了高性能 Java 压缩系统的设计与实现;第 4 节对本文设计的 Java 压缩系统进行了综合评估;第 5 节对本文进行了总结。

1　相关工作

压缩库是 Java 核心类库之一[4],是 Java 软件开发过程中经常用到的库,包含使用广泛的 deflate 算法[17],最流行的 Gzip 和 Zlib 压缩都使用该算法,二者在 Java 压缩库中均有较好的支持。但是 Gzip 和 Zlib 压缩无法满足复杂的生产环境中遇到的各种场景,因此,Apache commons compress 库[5]实现了各种常用压缩算法,包括 LZO[18]、LZ4[19]、BZip2[20]和 Snappy[21]等。Java 程序员可以根据实际的业务需求去使用适合的压缩 API,极大地提高了开发效率。

当前许多主流的大数据分布式处理框架如 Hadoop[22]、Spark[23]等都运行在 JVM 上,Hadoop 是使用 Java 语言开发的,Spark 是使用运行于 JVM 上的 Scala 语言开发的。在大数据的处理过程中,经常通过压缩来节省存储空间和提高网络传输效率。尽管压缩算法在不断改进,并且各种压缩算法在 Java 中已有很好的支持,但是压缩是计算密集型任务,纯软件压缩依然非常耗时。

随着登纳德缩放定律和摩尔定律的终结,近几年通过硬件加速器来提高 Java 应用性能的方式受到广泛关注。文献[24]设计了一个硬件垃圾收集(garbage collection,GC)加速器,将 JVM 中耗时的 GC 过程卸载到靠近内存的加速器中执行,有效降低 GC 时间,从而提高 Java 应用性能。Java OpenCL(JOCL)[25]使用 Java 语言对 OpenCL API 进行封装,使得 Java 应用程序可以使用图形处理器(graphics processing unit,GPU)来获得性能提升。

使用压缩加速器提高 Java 压缩性能已成为必然趋势,然而标准 JVM 不支持硬件加速器,Java 程序员不仅需要学习加速器的使用方法,还要了解加速器的底层知识,降低了开发效率。本文将 Java 应用程序调用压缩加速器的过程进行透明封装,降低了 Java 程序员使用本系统的编程难度。

压缩加速器可以有效提高压缩任务的执行效率,但是在 Java 应用中并不常用,主要原因除了编程难度较大以外,还因为目前压缩加速器大多采用松耦合的集成方式[6-11],存在较大的通信开销和数据搬运开销,导致加速器带来的性能收益有限。文献[12]指出了松耦合方式 CPU 与现场可编程门阵列(field programmable gate array,FPGA)之间通信开销大的问题,在 Intel-Altera HARPv2 实现了压缩加速器,但是其本质上依然是松耦合的集成方式。为了解决处理器与加速器之间通信开销大的问题,许多研究工作将处理器与加速器的距离拉近。IBM POWER 9 和 z15 处理器通过拉近与压缩加速器之间的距离[13],有效降低通信开销,从而提高压缩性能。本文使用领域前沿的 TCADer 框架集成紧耦合压缩加速器的处理器平台[15-16],处理器与加速器之间连接方式更加紧密,进一步降低通信开销,避免了大量的数据搬运,充分发挥加速器给 Java 压缩带来的性能优势。

2　背　景

2.1　Java 压缩

图 1 展示了 JVM 的核心模块以及使用 Java 压缩类库(如 Deflater API)的应用程序的基本执行过程[3]。首先应用程序会被 javac 编译器编译成平台无关的字节码文件;然后由 JVM 对字节码文件进行加载和执行。JVM 中的类加载器将应用程序编译后的字节码文件和所需的 Java 类库字节码文件加载到运行时数据区,并对类进行校验、解析和初始化。运行时数据区用于存放 JVM 加载的类信息、执行过程中使用的程序计数器、Java 堆栈和本地方法栈等重要数据。类加载完成后执行引擎开始解释执行或编译执行字节码。由于 Deflater 类的核心压缩算法是使用 C 语言实现的,所以在执行过程中需要

通过 Java 本地方法接口(Java native interface,JNI)来调用本地方法库(即 libzip. so),然后开始进行压缩。

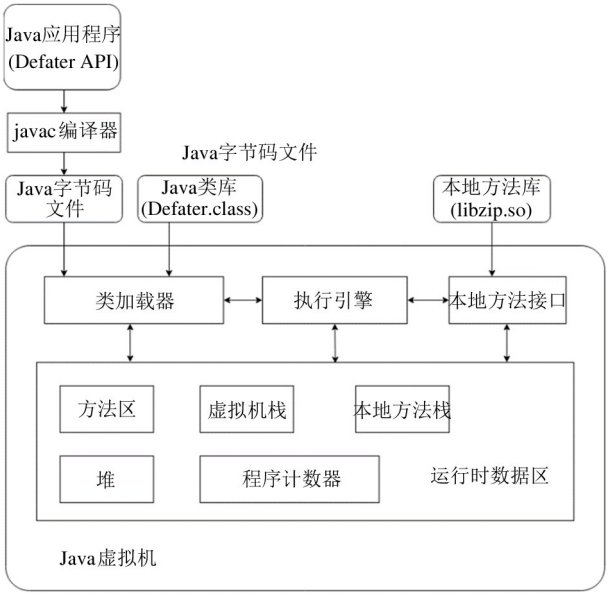

图 1　JVM 核心模块及应用程序执行过程

2.2　TCADer 框架

异构系统集成加速器的方式可以分为松耦合和紧耦合 2 类,松耦合方式 CPU 和加速器通过高速输入输出(input \ output,IO)总线或片上互连总线集成,紧耦合方式是在 CPU 流水线中添加硬件加速器。传统松耦合集成方式不适用于交互频繁或计算粒度小的加速器设计,紧耦合集成方式不适用于计算复杂的加速器设计。TCADer[15]是一种新型异构系统紧耦合加速器框架,它打破了传统异构系统对加速器设计的限制。使用 TCADer 框架,加速器的计算粒度和交互频率可以根据应用行为定制,不会受到集成和执行环境的制约,充分发挥出硬件加速器的算力。该框架中添加的协处理器管理单元和协处理器访存子系统简化了加速器设计,使 CPU 和加速器可以独立执行。该框架还使用轻量级用户态运行时环境和可重用协处理器指令实现了 CPU 和加速器之间的低延迟通信。在使用 TCADer 框架实现的异构系统中,加速器和 CPU 既具有松耦合结构灵活、独立的特点,又兼有紧耦合结构低通信延迟的性能优势。

2.3　紧耦合压缩加速器

本文研究的 Java 数据压缩主要是针对无损压缩,使用无损压缩后的数据可以精确还原成原始数据,做到零数据丢失。无损压缩算法中应用最广泛的是 Lempel-Ziv(LZ)算法家族,常用的 Gzip、LZ4、LZO 压缩等都属于这类算法,它们都是基于 LZ77 算法[26]衍生而来。因此本研究将基于核心的 LZ77 算法。

LZ77 算法由 Lempel 和 Ziv[26]于 1977 年提出,其主要概念是信息的时间局部性,即数据倾向于在不久的将来重复。算法核心是为当前输入找到最佳的历史匹配子串,从而在输出中通过保存子串的位置和长度,来达到保存整个子串的目的,以此节约大量空间。

本文使用的紧耦合数据压缩加速器实现了 LZ 类算法的核心 LZ77 算法,并且使用了脉动阵列结构[16];对比基于内容可寻址存储器(content addressable memories,CAM)的压缩加速器,基于脉动阵列的压缩加速器具有更低的硬件成本和更好的可测试性。现有压缩加速器大多位于片外,采用通过 PCIe(peripheral component interconnect express)总线连接的传统松耦合方式集成到 CPU,这种方式具有高延迟、数据通信开销大的缺点。而此压缩加速器使用 TCADer 紧耦合加速器框架集成到 CPU,可有效降低通信开销,实现和 CPU 的快速交互,并且 CPU 和压缩加速器共享末级缓存(last level cache,LLC),有效避免了数据搬运。因此,本文使用 TCADer 框架集成脉动阵列结构 LZ77 压缩加速器的处理器平台,设计了一个高性能 Java 压缩系统,以提高压缩速度、并降低 Java 程序员调用压缩加速器的编程难度。

3　高性能 Java 压缩系统

本文基于紧耦合压缩加速器设计了一个高性能 Java 压缩系统,为 Java 程序提供轻量级调用压缩加速器的运行时环境,兼具易编程、快速压缩的特点。

3.1　系统概述

本文提出的高性能 Java 压缩系统整体结构如

图 2 所示,主要由 3 部分构成:(1)Java LZ77 API;(2)用于 Java 压缩过程处理和加速器管理的 TCADer-Java 运行时;(3)使用 TACDer 框架集成了紧耦合压缩加速器的处理器平台。

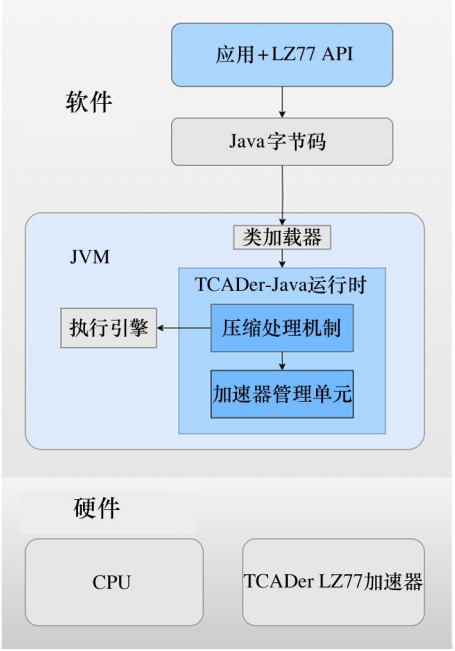

图 2　高性能 Java 压缩系统整体结构

本系统将 Java 应用程序调用压缩加速器的全部过程封装成 LZ77 API,使得复杂的压缩处理过程和加速器管理对编程人员透明,为其提供方便的高性能压缩方法接口,具体设计将在第 3.2 节中介绍。使用 LZ77 API 的 Java 应用程序代码会被编译为标准 Java 字节码,确保可以在标准 JVM 中运行,坚持 Java“一次编写,到处运行”的理念。

TCADer-Java 运行时是 JVM 中调用压缩 API 到加速器执行的必要过程,是支持 Java 透明使用压缩加速器并实现二者低延迟交互的重要保障。它包括压缩处理机制和加速器管理单元。压缩处理机制负责 LZ77 方法的识别和数据压缩的执行模式选择,并在确定使用加速器后进行数据和参数的准备。加速器管理单元会对 JVM 与加速器之间的交互进行管理,主要职责包括配置、启动加速器、接收加速器发来的中断信号及获取加速器执行结果。

本系统采用 TCADer 框架集成紧耦合压缩加速器的处理器平台,是 Java 应用程序高性能压缩的基础。使用硬件加速器替代原有复杂耗时的软件压缩操作来提升压缩速度。采用的 TCADer 框架避免了传统加速器通信延迟高和数据搬运开销大的问题,使得压缩加速器的使用不会为系统带来负作用。

3.2　Java LZ77 API

为了便于 Java 程序员使用压缩加速器,本系统对原 Java 压缩类库进行了扩展,将压缩加速器的调用过程封装成 LZ77 API。此 API 在设计上保持与原有 Deflater API 尽可能相似,以确保使用简单。LZ77 API 有 2 种执行模式:一种是直接使用 CPU 执行,另一种是使用压缩加速器来执行。这 2 种执行模式使用统一的编程接口,执行模式的选择取决于 TCADer-Java 运行时的分析结果。该过程对程序员透明,加速器的使用不会为其带来任何学习和使用的负担。

3.3　TCADer-Java 运行时

TCADer-Java 运行时是为 TCADer 框架集成压缩加速器的处理器平台设计的一套轻量级 Java 运行时环境,主要负责 Java 压缩过程处理和 JVM 与加速器之间交互的管理,具有通信延迟低和零数据搬运开销等特点。

本文提出的 TCADer-Java 运行时设计以及使用本系统的 Java 压缩执行过程如图 3 所示。首先,程序员使用 LZ77 API 来实现应用程序中的压缩部分;其次,此应用程序会被 javac 编译器编译成标准字节码文件,本系统没有增加任何设备特定字节码,这对维护 Java 跨平台特性非常重要。

编译后的字节码包含对 LZ77 方法的调用,TCADer-Java 运行时的压缩处理机制会在首次调用 LZ77 方法时进行识别,并且获取相应的压缩参数,如待压缩数据地址、长度和压缩后数据地址等信息。接着检查系统配置并确定数据压缩任务的执行模式。如果此时压缩加速器不可用,将采用纯软件压缩,即继续在标准 JVM 中执行压缩;如果压缩加速器可用,则向内核申请获取加速器使用权,开始为加速器准备数据和参数,使用加速器来执行高性能压缩。加速器与 CPU 共享地址空间,从而压缩处理机制使 JVM 和加速器以统一的方式分配和使用内存,在数据准备阶段只需要将 Java 数据压缩相关地址传递给加速器管理单元,省去了大量数据搬运和地

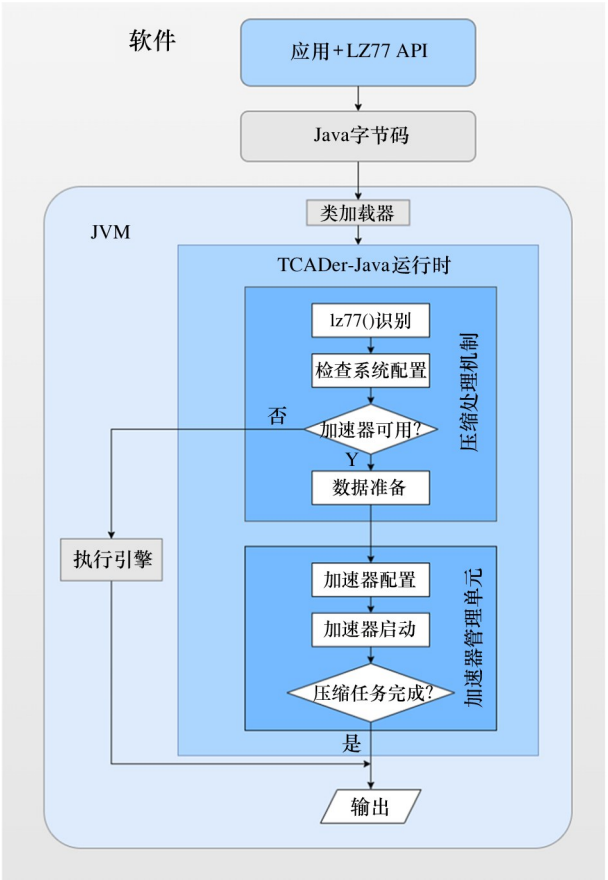

图 3　Java 压缩系统的执行过程

址空间转换开销,使得数据管理更加便捷高效。Java 压缩过程全部处理完成之后将交由加速器管理单元负责加速器相关控制。

加速器管理单元仅使用几条用户态指令快速配置和启动加速器,通信开销几乎为零。启动成功后压缩任务将在加速器上执行。当加速器压缩任务执行完成时,使用用户态中断通知机制来通知 JVM。JVM 收到加速器发来的用户态中断信号后,使用轻量级用户态中断处理跳到压缩后的指令位置。该过程全程在用户态完成,避免了传统加速器中断处理在用户态、核心态之间的切换开销,实现了 JVM 和加速器之间最快速的同步。

压缩处理机制仅在识别 LZ77 方法后使用压缩加速器,并且保证在压缩加速器不可用时依然可以正确执行压缩任务。加速器管理单元的功能基本都是在用户态完成的,对内核没有较大的改动。因此,TCADer-Java 运行时不会给系统带来安全性问题。

在压缩加速器执行过程中,JVM 可以并行执行与压缩无关的任务,待加速器压缩任务执行完毕后 JVM 再继续执行后续相关任务。该设计使得 JVM 执行线程与加速器上压缩任务的执行可以实现细粒度并行,以极小的通信开销充分发挥了 Java 压缩任务卸载到加速器的优势。

3.4　数据流分析

本系统通过定制硬件获取比软件高的计算性能以节省 CPU 时间,且采用 TCADer 紧耦合的加速器框架使得可以保持与原生软件方法一样的数据流通路,避免了传统加速器为系统带来的更多的带宽占用等负作用。图 4 展示了原生软件方法、传统压缩加速器和本系统分别在处理压缩任务时的数据流情况。

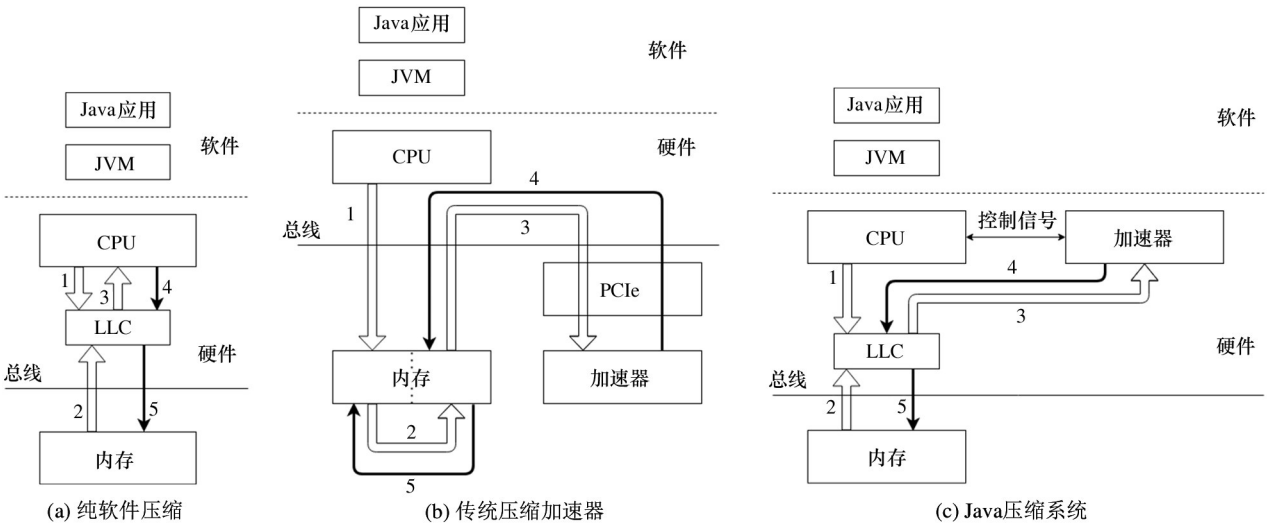

图 4　不同方式处理 Java 压缩任务时的数据流

目前,Java 普遍使用标准库或第三方库提供的纯软件方法进行数据压缩。纯软件压缩的数据流如图 4(a)所示,首先待压缩数据由 Java 应用执行过程中产生并存入 LLC(1)或从磁盘中读取到内存;然后从内存读到 LLC(2);接着 CPU 从 LLC 中读取待压缩数据进行压缩(3);待数据压缩完成之后,将压缩后的数据写到 LLC(4);最终写回到内存(5)。纯软件压缩的数据流很简单,但是压缩任务是计算密集型操作,会消耗大量 CPU 时间。

在一些研究或是专用领域会使用压缩加速器来提高压缩任务执行的性能。但通常传统松耦合方式的 CPU 和加速器之间不共享内存,因此两者之间大量的数据搬运开销严重制约了加速器带来的性能提升。并且压缩的数据量通常较大,在总线上的传输会占用大量的带宽,为系统带来负担。图 4(b)展示了 Java 压缩任务使用传统压缩加速器执行时的数据流。首先,待压缩数据由 Java 应用执行过程中产生并存入内存(1)或从磁盘中读取到内存,由于 CPU 和加速器地址空间独立,需要先将待压缩数据从 CPU 地址空间搬运到加速器地址空间(2);又由于传统松耦合加速器一般位于片外,需要将数据再通过 PCIe 总线搬运到加速器私有缓存(3);然后,加速器才能获取到数据开始执行压缩,而当压缩任务执行完成以后,又需要按照压缩前相反的顺序将压缩完的数据通过 PCIe 传回至内存(4)(5)。整个执行过程中存在繁重的数据搬运,并且大量的数据搬运还会占用宝贵的带宽,使得开销非常大。

本系统选择使用 TCADer 框架集成紧耦合压缩加速器,执行压缩任务的数据流如图 4(c)所示。首先,待压缩数据由 Java 应用执行过程中产生并存入 LLC(1)或从磁盘中读取到内存;然后从内存读到 LLC(2),由于 CPU 和加速器采用紧耦合方式,且共享 LLC 和地址空间,加速器在接收到 CPU 传来的配置和启动控制信号后,可以直接从 LLC 获取数据(3);执行压缩任务,在压缩任务执行完毕后,将压缩后的数据直接写到 LLC(4);最终写回到内存(5),CPU 接收到加速器发来任务完成的中断信号后便可获取到压缩后的数据。整个压缩过程的数据流和纯软件压缩基本一致,在节省大量 CPU 时间的同时,省去了大量数据搬运开销,充分发挥了加速器给 Java 压缩带来的性能优势。

4　实验分析

4.1　实验平台

为了评估本文提出的高性能 Java 压缩系统,将本系统运行在 Xilinx Virtex-7 FPGA VC 709 开发板上,使用开源的伯克利乱序处理器核(Berkeley Out-of-Order Machine, BOOM)[27] 作为 CPU 基本结构。BOOM 是加州大学伯克利分校使用硬件构造语言 Chisel[28] 开发的可综合、参数化的超标量、乱序 RISC-V 指令集架构处理器核,其配置参数见表 1,在 FPGA 上运行的主频为 70 MHz。压缩加速器使用 TCADer 框架集成到 BOOM。本系统使用的操作系统为 Linux kernel 5.10,软件部分 JavaLZ77 API 和 TCADer-Java 运行时在 OpenJDK 17[29] 中实现。本 Java 压缩系统可稳定运行于 FPGA 开发板上启动的 Linux 系统之上。为了降低实验误差,以下实验数据均为 10 次执行结果的平均值。

表 1　BOOM 配置参数

配置项	配置参数
取指宽度	4
译码宽度	2
指令重定序项数	64
访存单元发射宽度	1
定点单元发射宽度	2
ICache	16 kB
DCache	16 kB

实验将在同平台比较本文设计的 Java 压缩系统与 OpenJDK17 已有的主流软件压缩 Gzip 的压缩率和压缩速度 2 项重要指标。另外,实验还将本系统和 Java Gzip 软件压缩在 Intel Core i5-10400 处理器、主频 2.9 GHz 和 16 GB 内存的机器上执行的速度进行比较。

实验评估使用 Calgary corpus 和 Canterbury corpus 2 个无损压缩基准数据集,共 29 个文件。涵盖文本、二进制文件、图片、程序源码等多种文件类型,

文件大小范围为 3.6 kB ~ 1 MB，能够对本文提出的 Java 压缩系统进行全面有效的评估。

4.2　压缩加速比

设计实验统计了本 Java 压缩系统和纯软件 Java Gzip 压缩在 BOOM 和 Core i5 上的执行时间，然后将 Core i5 上的执行时间按 BOOM 进行同主频换算。为了更直观地展示本系统的性能提升，分别算出 3 种方式的压缩速度，以 Java Gzip 压缩在 BOOM 上的执行速度为基准，图 5 所示为 BOOM 上本 Java 压缩系统和 Core i5 上 Java Gzip 压缩速度分别相对于 BOOM 上 Java Gzip 压缩速度的加速比。由于本 Java 压缩系统对不同文件的性能提升存在较大的差别，为了清楚展示各个文件使用 Java 压缩系统获得的性能提升，图中 y 轴使用了对数刻度。从同平台 BOOM 上执行结果来看，压缩速度平均是软件 Java Gzip 压缩的 63 倍，最高可以达到 247 倍，最差情况速度也是 Java Gzip 的 2 倍。

本 Java 压缩系统与 Core i5 上执行的纯软件 Java Gzip 压缩相比，压缩速度平均是后者的 29 倍，最高可达 115 倍。除了 grammar.lsp 和 xargs.1 因为文件本身太小导致本 Java 压缩系统带来的性能提升不足以弥补 2 个处理器之间的性能差距、使得加速器效果没有发挥出来以外，对于其他文件，使用本 Java 压缩系统获得的性能提升均较为显著。

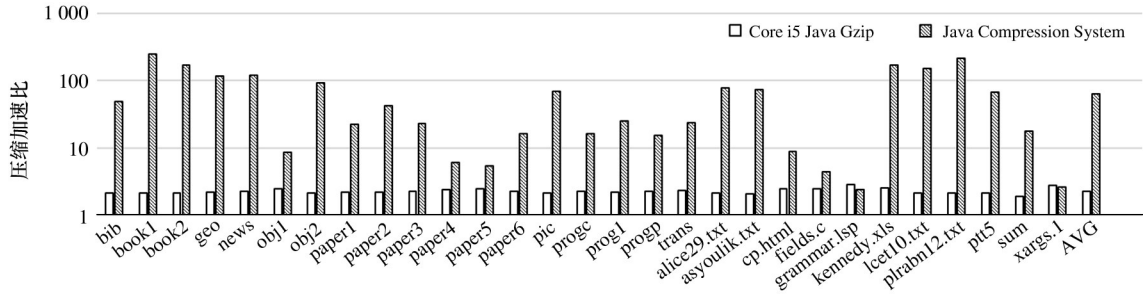

图 5　压缩加速比

4.3　吞吐率

图 6 展示了本 Java 压缩系统和同平台 BOOM 上执行 JavaGzip 的压缩吞吐率，其中 x 轴表示文件大小 *SIZE*(bytes)取 $\log_2$ 的值，图中的点对应 2 个基准数据集所有文件的吞吐率按文件大小从小到大排布的结果。可以明显看出，文件较大时本系统相对于软件 JavaGzip 压缩的性能提升更显著。JavaGzip 压缩吞吐率相对比较稳定，平均为 0.21 MB/s，最低 0.07 MB/s，最高 0.41 MB/s。本系统的压缩吞吐率随着文件大小的增加在不断升高，图中最低点压缩吞吐率为 0.35 MB/s，对应数据集中最小的文件 grammar.lsp，大小为 3.6 kB；最高点压缩吞吐率为 37.66 MB/s，对应数据集中最大的文件 kennedy.xls，大小为 1 MB。当文件大小超过 1 MB 后，压缩吞吐率可以超过 37.66 MB/s。

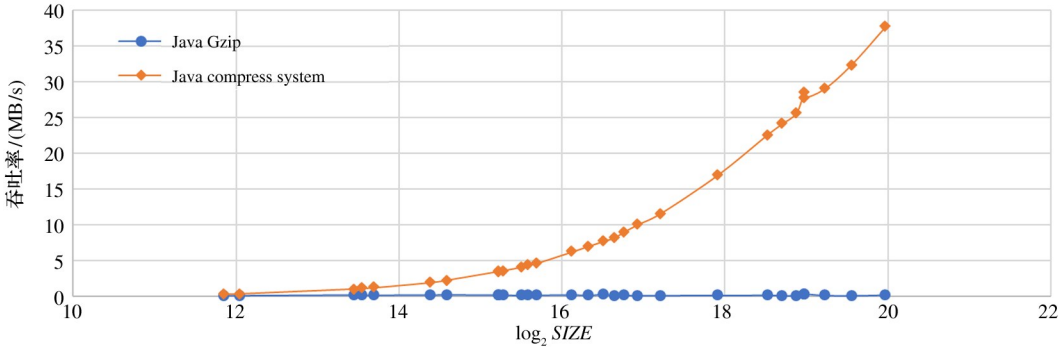

图 6　Java 压缩系统吞吐率

压缩吞吐率的计算公式为

$$Throughput = \frac{SIZE}{T_0 + T_{exe}} = \frac{1}{\frac{T_0}{SIZE} + \frac{T_{exe}}{SIZE}}$$

$$= \frac{1}{\frac{T_0}{SIZE} + \frac{1}{Throughput_{exe}}} \quad (1)$$

其中, $SIZE$ 为待压缩文件大小, T_0 为 2.1 节中分析的 Java 压缩过程中实际执行压缩以外的 JVM 执行时间, T_{exe} 为实际压缩执行时间, $Throughput_{exe}$ 为实际执行压缩的吞吐率。使用 JavaGzip 软件压缩时,即使压缩很小的文件也非常耗时,即实际压缩执行时间 T_{exe} 很大, T_0 可以忽略不记,此时根据式(1),JavaGzip 压缩的吞吐率几乎与实际执行压缩的吞吐率 $Throughput_{exe}$ 相等,而 $Throughput_{exe}$ 比较稳定,所以 JavaGzip 压缩的吞吐率也比较稳定。本系统使用的压缩加速器使实际执行压缩的吞吐率 $Throughput_{exe}$ 大幅提升[16],并且也比较稳定,所以实际压缩执行时间 T_{exe} 很小,导致 T_0 无法忽略,所以根据式(1)本系统压缩吞吐量会随着文件大小 $SIZE$ 的增加而不断提升,如图 6 所示。当文件增大到一定大小后,实际压缩执行时间 T_{exe} 增大到 T_0 可以忽略不记时,本系统的吞吐率将趋于压缩加速器的峰值吞吐率,因此,在大数据集下性能提升更显著。

4.4　压缩率

图 7 展示了使用纯软件 Java Gzip 压缩和本 Java 压缩系统压缩后节省的空间大小与原文件大小的百分比,百分比越高说明压缩后节省空间越多、压缩率越高。从图中可以看出,本压缩系统对基准数据集中 geo 以外的其他文件都有较好的压缩效果,压缩后 29 个文件平均节省 53.0% 的空间,效果最好的文件 pic 和 ptt5 空间节省了 83.0%,这表明本 Java 压缩系统对于大部分适合无损压缩的数据类型都适用。使用 Java Gzip 比本 Java 压缩系统的压缩效果稍好一些,对 29 个文件的压缩平均多节省 13.6% 的空间,其中压缩效果最差的文件也是 geo,这是因为 geo 文件中的数据是随机产生的,重复性不好,可获得的压缩效果有限。

本 Java 压缩系统比 Java Gzip 压缩率低是因为本系统使用的压缩加速器实现的是 LZ77 压缩算法,Gzip 压缩使用的 deflate 算法[17]是在 LZ77 算法基础上的改进,增加了 Huffman 编码来进一步提高压缩率。并且将复杂的软件压缩算法实现成硬件加速器来提高压缩速度的同时,会造成一定的压缩率损失[10-13]。本系统虽然压缩率稍低于 Java Gzip,但其整体压缩效果可观,且压缩速度远高于后者,并兼具易编程的特性。

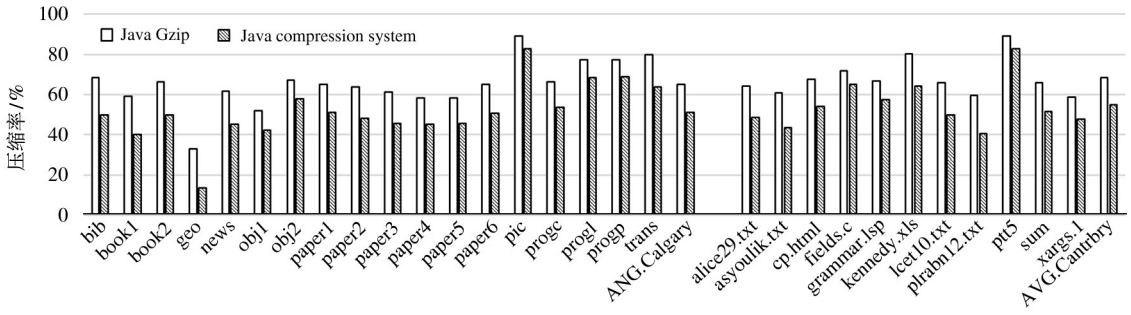

图 7　压缩率

5　结　论

随着大数据时代的到来,数据量爆炸式增长,数据压缩在数据高效的存储和传输中起着关键作用,因此对 Java 无损压缩的效率提出了更高要求。尽管无损压缩算法在 Java 类库中已有很好的支持,且算法在不断改进,但是纯软件压缩依然非常耗时。本文使用领域前沿的 TCADer 框架集成紧耦合压缩加速器的处理器平台,设计了一个高性能 Java 压缩系统。本系统将 Java 应用程序调用压缩加速器的过程进行了透明封装,使用时不会增加程序员的编程难度。轻量级 TCADer-Java 运行时环境可以有效减少通信开销、避免数据拷贝问题,充分发挥压缩加

速器带来的性能优势。实验结果表明,此系统使 Java 压缩性能大幅提升,压缩速度平均是软件 Java Gzip 压缩的 63 倍,压缩率适中,对于大部分适合无损压缩的数据类型都适用,且在大数据集下性能提升更显著。

参考文献

[1] NICOLAEB. High throughput data-compression for cloud storage[C]//Proceedings of the 3rd International Conference on Data Management in Grid and Peer-to-Peer Systems. Bilbao, Spain: Springer Link, 2010: 1-12.

[2] PÖSS M, POTAPOV D. Data compression in oracle[C]//Proceedings of the 2003 VLDB Conference. Berlin, Germany:Morgan Kaufmann, 2003:937-947.

[3] LINDHOLM T, YELLIN F, BRACHA G, et al. The Java virtual machine specification, Java SE 8 edition[M]. Boston: Addison-Wedey Professional, 2014.

[4] Package java. util. zip[EB/OL]. [2023-03-30]. https://docs. oracle. com/en/java/javase/17/docs/api/java. base/java/util/zip/package-summary. html.

[5] APACHE. Apache Commons Compress[EB/OL]. [2023-03-30]. https://commons. apache. org/proper/commons-compress/.

[6] BARTÍK M, UBIK S, KUBALIK P. LZ4 compression algorithm on FPGA[C]//2015 IEEE International Conference on Electronics, Circuits, and Systems. Cairo, Egypt: IEEE, 2015:179-182.

[7] FOWERS J, KIM J Y, BURGER D, et al. A scalable high-bandwidth architecture for lossless compression on FPGAs[C]//2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines. Vancouver, Canada: IEEE, 2015:52-59.

[8] SOFIA A T, LEWIS C C, JACOBI C, et al. Integrated high-performance data compression in the IBM Z13[J]. IBM Journal of Research and Development, 2015,59(4-5):1- 10.

[9] FOWERS J, KIM J Y, BURGERD, et al. A scalable high-bandwidth architecture for lossless compression on FPGAs [C]//2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines. Vancouver, Canada:IEEE, 2015:52-59.

[10] 李冰, 王超凡, 顾巍, 等. Gzip 压缩的硬件加速电路设计[J]. 电子学报, 2017,45(3):540.

[11] CHEN J, DAVERVELDT M, AL-ARS Z. FPGA acceleration of ZSTD compression algorithm[C]//2021 IEEE International Parallel and Distributed Processing Symposium Workshops. Portland, USA: IEEE, 2021:188-191.

[12] QIAO W, DU J, FANG Z, et al. High-throughput lossless compression on tightly coupled CPU-FPGA platforms [C]//2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines. Boulder, USA: IEEE, 2018:37-44.

[13] ABALI B, BLANER B, REILLY J, et al. Data compression accelerator on IBM POWER9 and Z15 processors: industrial product[C]//2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA). Valencia, Spain: IEEE, 2020: 1-14.

[14] COTA E G, MANTOVANI P, DI GUGLIELMO G, et al. An analysis of accelerator coupling in heterogeneous architectures[C]//Proceedings of the 52nd Annual Design Automation Conference. San Francisco, USA: IEEE, 2015:1-6.

[15] LI W, LIU T, XIAO Z, et al. TCADer: a tightly coupled accelerator design framework for heterogeneous system with hardware/software co-design[J]. Journal of Systems Architecture, 2023,136:102822.

[16] 肖子原. 紧耦合的数据压缩加速器[D]. 北京:中国科学院大学, 2022.

[17] DEUTSCH P. RFC 1951: deflate compressed data format specification version 1. 3[R]. Aladdin: Aladdin Enterprises, 1996.

[18] OBERHUMER M. LZO: a real-time data compression library[EB/OL]. [2023-03-23]. http://www. oberhumer. com/opensource/lzo.

[19] COLLET Y. LZ4: extremely fast compression algorithm [EB/OL]. [2023-03-23]. http://lz4. github. io/lz4.

[20] SEWARD J. Bzip2[EB/OL]. [2023-03-23]. https://gitlab. com/bzip2/bzip2.

[21] DEAN J, GHEMAWAT S, GUNDERSON S H. Snappy, a fast compressor/decompressor [EB/OL]. [2023-03-23]. https://github. com/google/snappy.

[22] APACHE. Apache Hadoop[EB/OL]. [2023-03-23]. http://hadoop. apache. org/.

[23] ZAHARIA M, CHOWDHURY M, FRANKLIN M J, et al. Spark: cluster computing with working sets[J]. Hot-

Cloud, 2010,10(10):95.

[24] MAAS M, ASANOVI Ć K, KUBIATOWICZ J. A hardware accelerator for tracing garbage collection[C]//2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture(ISCA). Los Angeles, USA: IEEE, 2018:138-151.

[25] GitHub. Java bindings for OpenCL[EB/OL]. [2023-03-23]. http://www.jocl.org, 2017.

[26] ZIV J, LEMPEL A. A universal algorithm for sequential data compression[J]. IEEE Transactions on information theory, 1977,23(3):337-343.

[27] AMID A, BIANCOLIN D, GONZALEZ A, et al. Chipyard: integrated design, simulation, and implementation framework for custom SOCS[J]. IEEE Micro, 2020,40(4):10-21.

[28] BACHRACH J, VO H, RICHARDS B, et al. Chisel: constructing hardware in a scala embedded language[C]// Proceedings of the 49th Annual Design Automation Conference. San Francisco, USA: IEEE, 2012: 1216-1225.

[29] OpenJDK. OpenJDK17[EB/OL]. [2023-03-23]. https://openjdk.org/projects/jdk/17.

High-performance Java compression system based on tightly coupled accelerator

WANG Xue[*,**], LI Wenqing[*,**], ZHANG Tingting[***], ZHANG Fuxin[*,**], WANG Jian[*,**], AO Qi[***]

([*]State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([**]University of Chinese Academy of Sciences, Beijing 100049)

([***]Loongson Technology Corporation Limited, Beijing 100190)

Abstract

Java lossless compression is becoming pervasive in a broad range of software systems. Although the compression algorithm has been improved, the low compression speed of Java lossless compression is still a problem. To solve this problem, a high-performance java compression system is proposed. The proposed system adopts the TCADer framework to integrate the CPU with a tightly coupled compression accelerator. Also, the packaging of the compression accelerator is implemented inside the Java virtual machine (JVM). Furthermore, a lightweight runtime environment is also provided for hardware management. In this way, This system can effectively reduce communication overhead and avoid data copy costs, and has the characteristics of easy programming. Experimental results show that this system can greatly improve the performance of Java compression with an average of 63 × (up to 247 ×) speedup over the mainstream Java Gzip compression. Moreover, this system also shows good performance with larger files.

Key words: Java compression, lossless compression, tightly coupled accelerator, Java virtual machine (JVM)