DOI:10.20079/j.issn.1001-893x.220917001

SpMV 计算的 ARM 和 FPGA 异构加速器设计*

朱明达,薛济擎,艾纯瑶

(中国石油大学(北京)信息科学与工程学院,北京 102249)

摘　要:针对稀疏矩阵向量乘(Sparse Matrix-Vector Multiplication,SpMV)在边缘端实施效率不高的问题,以稀疏矩阵的存储格式、SpMV 的现场可编程门阵列(Field Programmable Gate Array,FPGA)加速为研究对象,提出了一种多端口改进的行压缩存储格式(Modified Compressed Sparse Row Format,MCSR)与 ARM+FPGA 架构任务级数据级硬件优化相结合的加速方法。使用多个端口并行存取数据来提高计算并行度;使用数据流、循环流水实现循环间、循环内的并行加速;使用数组分割、流传输实现数据的细粒度并行缓存与计算;使用 ARM+FPGA 架构,ARM 完成对系统的控制,将计算卸载到 FPGA 并行加速。实验结果表明,并行加速优化后的 ARM+FPGA 方案相较于单 ARM 方案最高可达 10 倍的加速效果,而且增加的资源消耗在可接受范围内,矩阵规模越大非零值越多加速效果越明显。研究成果在边缘端实施 SpMV 计算方面有一定实用价值。

关键词:稀疏矩阵向量乘(SpMV);异构加速器;硬件加速

开放科学(资源服务)标识码(OSID):

中图分类号:TP332.2　　**文献标志码**:A　　**文章编号**:1001-893X(2024)02-0302-08

Design of an ARM and FPGA Heterogeneous Accelerator for SpMV Computation

ZHU Mingda, XUE Jiqing, AI Chunyao

(College of Information Science and Engineering, China University of Petroleum, Beijing 102249, China)

Abstract: To address the problem of inefficient implementation of sparse matrix-vector multiplication (SpMV) at the edge, the authors study the storage format of sparse matrix and field programmable gate array (FPGA) acceleration method of SpMV and propose a multi-port modified compressed row format (MCSR) acceleration method combined with task-level data-level hardware optimization in ARM+FPGA architecture. Computational parallelism is improved by using multiple ports to access data in parallel. Parallel acceleration between and within loops is achieved using dataflow and pipeline. Fine-grained parallel caching and computation of data is achieved using array partition and stream transfer. The ARM+FPGA architecture is used, with ARM completing the control of the system and offloading the computation to the FPGA for parallel acceleration. Experimental results show that the parallel acceleration optimized ARM+FPGA scheme can achieve up to 10 times acceleration compared with the single ARM scheme. And the increased resource consumption is within the acceptable range. The results also show that the larger the matrix size, the more non-zero value, the more obvious the acceleration effect. The research results are of practical value in the implementation of SpMV computing at the edge.

Key words: sparse matrix-vector multiplication (SpMV); heterogeneous accelerator; hardware acceleration

*　**收稿日期**:2022-09-17;**修回日期**:2022-11-04
项目基金:中国高校产学研创新基金(2020HYA08001);中国石油大学(北京)科研基金(2462020YXZZ025)
通信作者:朱明达　　Email:zhumingda@cup.edu.cn

0　引　言

随着人工智能技术的快速发展,算力向前端及边缘端迁移的趋势愈加明显[1]。由于功耗、体积方面的限制,ARM(Advanced RISC Machine)、FPGA(Field Programmable Gate Array)成为了边缘端算力实现的重要载体。稀疏矩阵向量乘(Sparse Matrix-Vector Multiplication,SpMV)常用于低延迟和高吞吐量的数据分析工作,除了在许多科学计算应用中占据主导地位,在边缘数据分析处理中也越发重要。然而,SpMV 的性能受到可用存储带宽的限制,ARM 的串行计算特点只能维持峰值计算性能的一小部分,这使得 SpMV 在边缘端有效实现极具挑战性。

近年来,许多学者开展了对 SpMV 运算加速的研究。文献[2]提出了一种基于高带宽内存的数据流 SpMV 加速器,克服了大矩阵存储的问题;还提出了两步流处理方法,但导致了 FPGA 资源开销非常高。文献[3-4]提出的 FPGA 加速方法通用性相对不足。文献[5-8]提出的方法基于 CPU 或 GPU 来实现[9-11],功耗较高,不利于在边缘端实现。在实现平台上,对比 CPU 及 GPU,FPGA 可以为不容易并行化的结构提供位级并行计算[12],具有高并行性、强控制能力和可重构性,逐渐成为加速计算的重要解决方案,是边缘端 ARM 方案的重要补充。

本文基于异构 Zynq 系列 FPGA 设计了一种多端口、高吞吐量的 SpMV 存储格式与加速系统:①设计了多端口改进的行压缩存储格式(Modified Compressed Sparse Row Format,MCSR)存储稀疏矩阵,提高并行度与计算效率;②研究了在嵌入式异构平台上加速算法的实现,在 FPGA 上完成对算法的加速,使用 ARM 完成对系统的控制,在 ARM+FPGA 的架构上利用软硬件协同方式对加速方法进行了验证;③ARM + FPGA 并行加速后的设计相比于单 ARM 方案可以达到 10 倍的加速效果,计算效率显著提升,特别是矩阵规模越大非零值越多加速效果越明显。

1　加速方法

1.1　稀疏矩阵的存储格式

为了减少稀疏矩阵的存储空间,避免零值参与产生无意义的计算,一般采用特定的存储格式,常用的有三元组存储格式(Coordinate Format,COO)和行压缩存储格式(Compressed Sparse Row Format,CSR)[13-14]。目前稀疏矩阵较常用、效率也较高的存储格式是 CSR 格式,但作为一种针对矩阵的通用方法,CSR 格式在基于 FPGA 实现时并不是最高效的。由于 CSR 格式中 row_ptr 行偏移数组存储的是行起始偏移位置,每行非零元素个数需要在迭代过程中才能确定,因此外循环不能使用循环流水优化方法,必须按顺序执行。这使得 SpMV 计算效率低下,迭代延迟很高。为了解决这个问题,本文使用 MCSR 存储格式,并在此基础上进行改进。

如图 1(a)所示,MCSR 格式中的 col_index、value 数组与 CSR 格式相同,不同之处在于它将 CSR 格式的 row_ptr 行偏移数组更改为 row_len 行长度数组,即存储每行非零元素的个数。结合 FPGA 的特点,为了使用多端口计算进行硬件优化,本文先将 MCSR 格式行、列索引合并起来,形成新的 indices 索引数组,再将 indices 和 value 数组按行分成 p 部分。合并后的单端口 MCSR 格式如图 1(b)所示,通过连接 row_ptr 和对应数量的 col_index 来定义。行、列索引的交错合并过程不包括任何计算,可在边缘计算平台上处理,也可在本地预处理。在此基础上,将单端口 MCSR 格式进一步优化为多端口 MCSR 格式,以便利用多个端口并行存取数据来提高算法性能。通过将输入的稀疏矩阵按行分成 p 部分,每一部分单独处理,来实现 p 倍的运行加速。图 1(c)展示了当 p 为 2 时 indices 和 value 数组的分块结果。具体分块的 p 取值将在多端口计算方法中详细阐述。

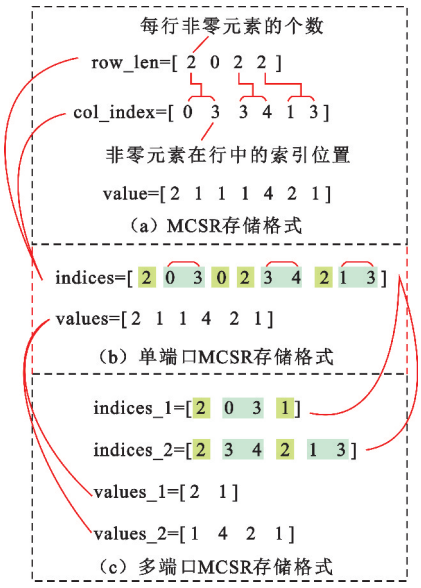


图 1　MCSR 存储格式

Fig. 1 Schematic diagram of MCSR storage format

1.2　硬件优化方法

针对 SpMV 的运算特点,本文将硬件优化方法分为两类:任务级优化和数据级优化。任务级优化

包括多端口计算、数据流、循环流水。在使用优化的多端口 MCSR 存储格式时,可使用数据流实现多端口任务并行,使用循环流水实现每个端口内的任务并行。数据级优化包括数组分割和流传输。数组分割实现了数据读写的并行,流传输实现了数据的缓存。

1.2.1 多端口计算

SpMV 计算包含读、计算和写 3 个任务,为了提高读、写任务的数据吞吐量,本文利用多个端口并行地将数据从存储器传输到 FPGA。如果 FPGA 包含 D 个存储器端口,每个端口具有 B 位,稀疏矩阵 value、indices 中的每个元素分别为 g、h 比特,可将输入稀疏矩阵的行分成 p 部分对应多端口 MCSR 的 p 个端口,每一部分都单独处理,从而最大限度提高 p 倍运行速度。p 计算公式如式(1)所示:

$$p \leqslant \frac{D \times B}{g+h} \tag{1}$$

每个硬件线程计算输出向量 y 的一部分,最后将 y 合并保存到 BRAM 中并传输到 DDR 中。增大 p 值可有效减小 SpMV 计算延时,但会增大资源利用率。综合考虑延时与硬件资源后,本文选择端口数 $p=8$ 进行实验。

1.2.2 数据流

数据流优化方法实现任务级流水线,允许函数或循环之间的操作并行执行,可减少时延,增加 RTL 并发度,提高总体吞吐量。不使用数据流优化时,所有操作默认顺序执行。例如,程序必须在访问数组的函数或循环之前完成对数组的所有读、写访问。这严重限制了下一个使用数组的函数或循环的启动操作。使用数据流优化后,当前函数或循环中的操作可在上一个函数或循环完成其所有操作之前启动,如图 2 所示。

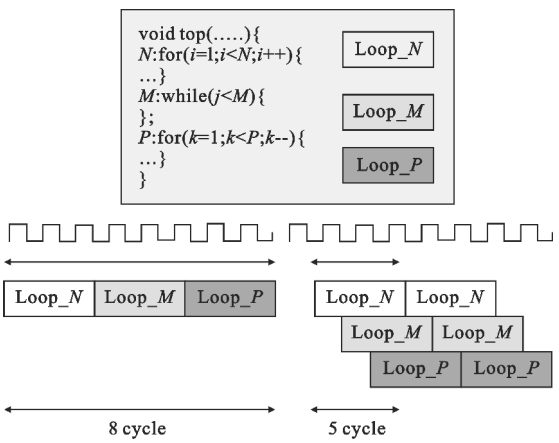


图 2 数据流优化方法

Fig. 2 Data flow optimization method

如图 3 所示,顺序函数之间具有数据流,在循环之间插入 Channel 通道(Ping-pong RAM、FIFO 或 Register),以确保数据可从一个循环异步流到下一个循环。

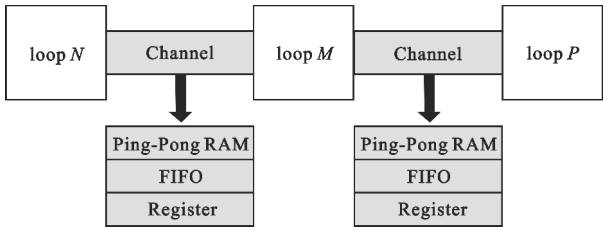


图 3 数据流优化方法细节

Fig. 3 Details of data flow optimization method

在使用优化的多端口 MCSR 存储格式计算 SpMV 时,每个端口之间数据不存在相互关联,因此可使用数据流实现多端口任务并行。其中可并行的任务如下:多个端口 indices 和 values 数据的并行读入、缓存;多个端口 indices 数据中 rows 和 cols 的并行缓存;多个端口待计算值(稀疏矩阵中某一元素)与列向量对应位的累加求和;多个端口的 results 数据的并行缓存、输出。具体数据缓存类型的选择将在后文流传输方法中阐述。

1.2.3 循环流水

针对单个循环的优化,一般采用循环流水,例如在 for 循环中对循环下的代码做流水化处理。当输入的数据不存在依赖性时,可并行化对数据进行读、计算和写操作,此时可将任务间隔优化到 1,相当于一个时钟周期一个运算单元可完成一个操作。循环流水方法允许以并行方式实现循环操作,如图 4 所示。

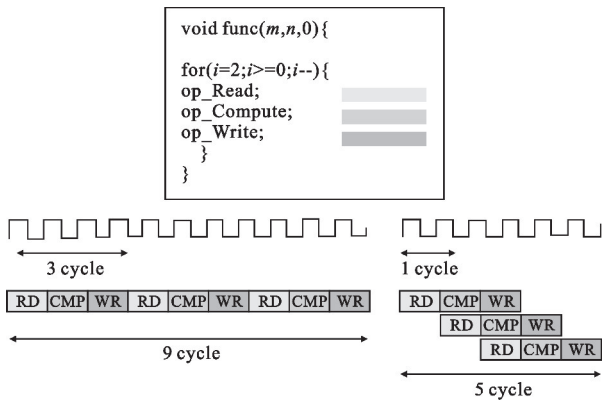


图 4 循环流水优化方法

Fig. 4 Optimization method of circulation flow

SpMV 运算中数据是顺序流动的,每个端口内

数据不存在相互关联,因此可使用循环流水实现每个端口内的任务并行。其中可并行的任务如下:每个端口 indices 和 values 数据的并行读入、缓存;每个端口 indices 数据中 rows 和 cols 的并行缓存;每个端口待计算值(稀疏矩阵中某一元素)与列向量对应位乘积的累加求和;每个端口的 results 数据的并行缓存、输出。

1.2.4　数组分割

在 FPGA 中,每个 BRAM 都有两个共享数据的可配置独立端口,用于片上数据缓存、FIFO 缓冲。默认情况下数组中 N 个元素连续存储在一块 BRAM 中,则一个时钟周期只能提供一个数据。为了最优化数据级流水线间隔,使用数组分割方法将原始数组(单一 BRAM)拆分为多个更小的数组(多个 BRAM)。这可有效增加内存的读写端口数量,改善设计吞吐量,提高数据读写并行度。

在每个端口待计算值(稀疏矩阵中的某一非零值)与列向量对应位乘积的任务中,默认情况一个端口中的 indices 与 value 数组分别连续存储在两块 BRAM 中。BRAM 最大的端口数量为 2,严重限制了读和写的吞吐量。因此为了改善带宽,使用数组分割方法将原一维数组按独立元素进行拆分,将内存分解为单个寄存器,有效实现了数据读写的并行。

1.2.5　流传输

流传输优化方法决定了数据流通道中的数据缓存类型。如图 3 所示,在数据流的函数主体或循环主体独立通道,将每项任务的结果缓存在通道中。根据不同任务可选择不同的数据缓存类型,如 Ping-Pong RAM 或 FIFO。通常情况下,Ping-Pong RAM 用来存取矢量数组,FIFO 用来存取标量。数据流所添加的每个通道都包含用于指示 Ping-Pong RAM 或 FIFO 已满或已空的信号。流传输优化方法如图 5 所示。流传输方法的优势在于不需要地址管理,不足是读取完当前数据之后无法再次对该数据进行读取。但是由于 SpMV 运算只涉及数据的顺序读写,不受该不足影响,因此可使用流传输方法将数据流通道中的数据缓存类型配置为 FIFO,可有效减小资源利用率,提高数据存取并行度。

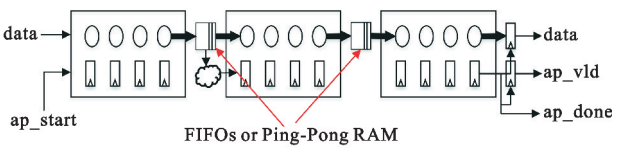


图 5　流传输优化方法细节

Fig. 5 Details of stream transmission optimization method

2　硬件实现

本文采用 Xilinx ZCU102 开发平台,基于 Xilinx Zynq UltraScale+ XCZU9EG-2FFVB1156E MPSoC 芯片进行了实现与测试[15]。

基于 ZCU102 平台的加速结构如图 6 所示。AXI Interconnect 为 AXI 总线实现 PS 端与 PL 端互联,Data Mover 为数据搬运器,SpMV accelerator 为稀疏矩阵乘法加速器。SD 卡存入 Linux 系统硬件启动文件与数据,程序初始化时 PS 端将 SD 卡数据读入 DDR4,PL 端 SpMV 加速模块通过直接内存存取方式(Direct Memory Access,DMA)获取 DDR4 中的数据,PL 端实现 SpMV 并行计算。

本文采用 Xilinx 的 SDSoC 开发环境,将. cpp 文件分别转换为 BOOT. bin、image. ub、. elf 文件和. bit 文件。其中,BOOT. bin 为启动文件,包含引导加载程序(FSBL)、引导程序(U-Boot);image. ub 包含 Linux 启动映像;. elf 为应用程序的二进制可执行文件,可在 Linux 系统中调用相关软硬件资源,完成数据的读取、输出和运算;. bit 为 FPGA 比特流文件。图 6 给出了基于 ARM+FGGA 的 SpMV 算法加速结构。

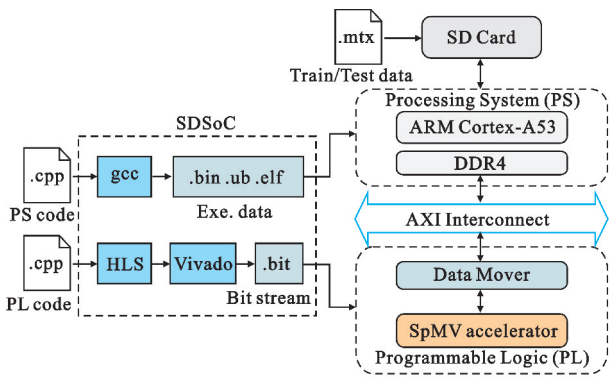


图 6　基于 ARM+FPGA 的 SpMV 算法加速结构

Fig. 6 Acceleration structure of SpMV based on ARM+FPGA

在硬件设计中,数据传输接口的设计尤为重要。本文中,由于传输稀疏矩阵非零值与计算结果时涉及较大数据量,因此选用 AXI MM 作为 Data Mover 传输数据,使用 SDS data mem attribute 指令约束数据存储地址连续性。相比 AXI GP 与 AXI ACP,AXI HP 接口有更高的读写带宽,且 AXI MM Data Mover 要求 PL 端口为主端口,PS 端口为从端口,因此选用 AXI HP 接口,使用 SDS data zero_copy 指令约束数据传输量。由于 Random 接口实现数据随机读取会占用较多的 BRAM 等片上资源,且 SpMV 计算过程中数据是顺序读取或写入的,因此选用 Sequential

接口,使用 SDS data access pattern 指令约束硬件函数数据访问方式。

根据本文提出的加速优化方法对 SpMV 进行硬件优化,结合以上加速结构,可得到优化后的 SpMV 程序分析结果,如图 7 所示,可以看出程序已经实现了多端口 indices、values 数据的并行读入、缓存,和多端口 results 数据的缓存、输出。图中,fmul 为程序中的乘积操作,fadd 为乘积结果与缓存 results 值的累加求和操作,已经实现了乘积与累加求和的并行操作。

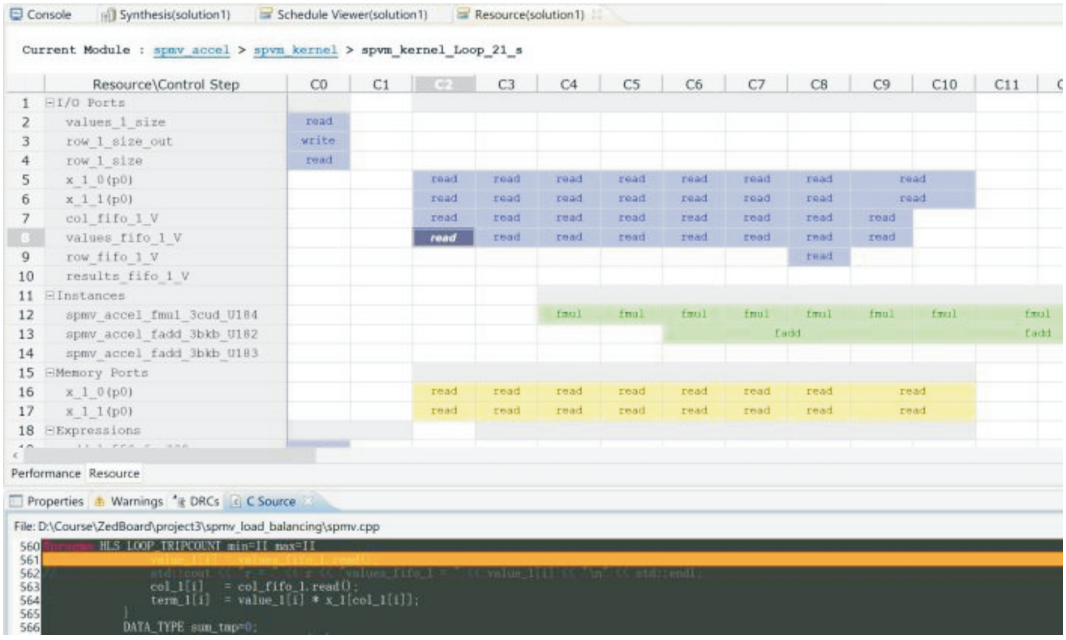


图 7 SpMV 程序分析

Fig. 7 SpMV program analysis

3 结果分析

3.1 数据集介绍

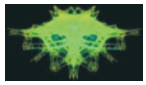
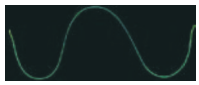
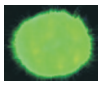
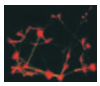
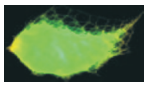
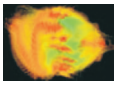
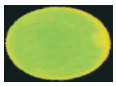
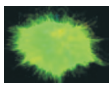
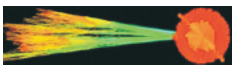
佛罗里达大学(University of Florida)稀疏矩阵库包含大量从应用程序中收集的稀疏矩阵,被广泛应用于基准测试。本文从中选择 19 个不同类型的稀疏矩阵,行列数范围在 $10^2 \sim 10^5$,非零值数量在 $10^2 \sim 10^6$。在计算 SpMV 时,稀疏矩阵是所选测试矩阵,密集向量是 rand( )函数产生的随机数。rand( )%($b-a+1$)+a 可以产生[a,b]范围的一个随机整数,本文选取 $a=0$,$b=1$ 做测试。稀疏矩阵的属性包含行数 m、列数 n、非零值、稀疏度、2D 图、3D 图,如表 1 所示。

表 1 稀疏矩阵属性

Tab. 1 Sparse matrix attributes

稀疏矩阵	行数 m	列数 n	非零值	稀疏度/%	2D 图	3D 图
bcsstk03	112	112	376	2. 99		
rotor1	100	100	708	7. 08		
bcsstk05	153	153	2 423	10. 35		
fpga_dcop_11	1 220	1 220	5 892	0. 40		
spaceStation_5	1 020	1 020	7 859	0. 76		

Tab. 1(Continuation)

稀疏矩阵	行数 m	列数 n	非零值	稀疏度/%	2D 图	3D 图
cage8	1 016	1 016	11 003	1.07		
cage9	3 534	3 534	41 594	0.333		
c-48	18 354	18 354	92 217	0.03		
mhd4800a	4 800	4 800	102 252	0.44		
abtaha2	37 932	332	137 228	1.09		
rajat22	39 900	39 900	197 264	0.01		
TF16	15 437	19 321	216 173	0.07		
g7jac080	23 672	23 672	293 976	0.05		
Sio	33 404	33 404	675 528	0.06		
lhr34c	35 152	35 152	764 014	0.39		
scircuit	170 998	170 998	958 936	0.01		
IG5-17	30 162	27 944	103 5008	0.12		
mixtank_new	29 960	29 960	1 995 041	0.22		
TSOPF_RS	28 338	28 338	2 943 887	0.37		

3.2　性能对比

为了验证系统性能,本文分别完成了基于 PS 端(即 ARM)和基于 PS+PL(即 ARM+FPGA)的系统设计。基于 PS 的设计是只使用 ARM 处理器系统完成全部计算,而基于 PS+PL 的设计将 SpMV 移植到 PL 端完成,由 FPGA 实现 SpMV 计算加速。

在 ARM+FPGA 系统设计中,又分别实现了 SpMV 并行加速前与并行加速后两种设计。SpMV 并行加速前采用原始的 CSR 存储格式,并行加速后采用本文的多端口 MCSR 格式+数据流+循环流水+数组分割+流传输的硬件优化方法。这一对比中变量为加速方法,不变量为测试的稀疏矩阵和 FPGA 实现平台。通过对比并行加速前后 SpMV 的硬件资源利用率、处理所需时间等性能指标,评价 SpMV 的硬件优化效果。

因此,本文一共实现了 3 种系统设计方式进行

对比,分别是“单 ARM”方案、“ARM+FPGA 并行加速前”方案、“ARM+FPGA 并行加速后”方案。分别对所选的 19 个稀疏矩阵进行 SpMV 计算对比,运行时间如图 8 所示,“ARM+FPGA 并行加速后”相对于“单 ARM”的加速比如表 2 所示。可见当稀疏矩阵规模较小非零值较少时“单 ARM”的计算效率较高,但是当规模变大非零值少量增加就会导致 ARM 计算效率下降。而随着矩阵规模增大非零值增加,本文提出的“ARM+FPGA 并行加速后”方案加速效果显著提高。

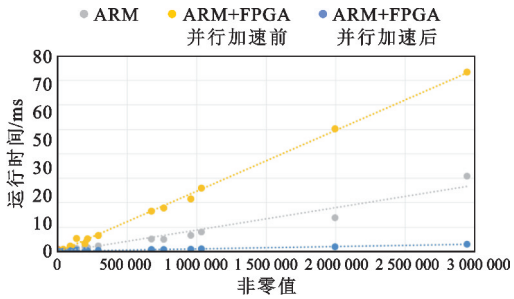


图 8 SpMV 计算运行时间

Fig. 8 SpMV calculation running time

表 2 SpMV 计算核心横向对比结果

Tab. 2 Horizontal comparison results of SpMV calculation core

稀疏矩阵	运算时间/ms			ARM+FPGA 并行加速后对单 ARM 加速比
	单 ARM	ARM+FPGA 并行加速前	ARM+FPGA 并行加速后	
bcsstk03	0.005 319	0.016 8	0.016 5	0.32
rotor1	0.007 779	0.030 4	0.024 7	0.31
bcsstk05	0.031 272	0.064 3	0.029 5	1.06
fpga_dcop_11	0.050 811	0.148 8	0.033 6	1.51
spaceStation_5	0.073 115	0.176 1	0.034 8	2.10
cage8	0.078 148	0.238 3	0.035 2	2.22
cage9	0.508 012	0.933 3	0.142 7	3.56
c-48	1.233 822	2.217 9	0.303 4	4.07
mhd4800a	0.575 088	1.679 4	0.139 6	4.12
abtaha2	1.289 203	5.321 4	0.586 7	2.20
rajat22	1.957 403	3.258 0	0.334 5	5.85
TF16	1.677 043	5.171 9	0.334 1	5.02
g7jac080	2.297 791	6.557 7	0.441 3	5.21
SiO	5.098 102	16.493 9	0.861 3	5.92
lhr34c	4.973 272	17.790 8	0.848 8	5.86
scircuit	6.629 088	21.501 6	0.955 2	6.94
IG5-17	7.994 850	25.906 2	1.110 9	7.20
mixtank_new	13.812 478	50.144 1	1.955 7	7.06
TSOPF_RS	30.806 370	73.329 1	2.972 4	10.36

根据 FPGA 开发软件 Vivado 生成的资源使用报表可以看出,优化前后 FPGA 的核心资源利用率有所增加,增加的资源消耗在可接受范围内,如表 3 所示。优化后实现了并行加速,各硬件资源利用率相比优化前略有增加。

表 3 并行加速前后 FPGA 核心资源使用对比

Tab. 3 Comparison of FPGA core resource usage before and after parallel acceleration

核心资源	总资源量	并行加速前使用量	并行加速前使用占比/%	并行加速后使用量	并行加速后使用占比/%
BRAM	1 824	101	5.54	312	17.11
DSP	2 520	5	0.20	56	2.22
FF	548 160	4 460	0.81	31 778	5.80
LUT	274 080	5 439	1.98	39 643	14.46

4 结束语

本文针对 FPGA 的 SpMV 计算加速问题提出了一种多端口的 MCSR 存储格式,结合数据流、循环流水、数组分割、流传输等加速优化方法实现了 SpMV 加速求解器设计。对比研究了“单 ARM”“ARM+FPGA 并行加速前”“ARM+FPGA 并行加速后”3 种 SpMV 的计算耗时与资源使用,实验结果表明,本文设计的加速器在稀疏矩阵规模增大非零值越多的情况下加速效果越明显。在本文所实验的稀疏矩阵中“ARM+FPGA 并行加速后”相较于“单 ARM”方案最高可以达到 10 倍的加速效果,而且增加的资源消耗在可接受范围内。因此,本文方法在边缘端实施 SpMV 计算方面具有一定的优势。

本文为了保证 3 种对比方案数据的一致性,都使用了浮点数计算,虽保证了计算准确率,但增加了 FPGA 硬件资源消耗。后续工作计划采用浮点数转定点数的方法,争取用有限的准确率损失换取硬件资源利用率的减少。

参考文献:

[1] 刘斐,曹钰杰,章国安. 车联网场景下移动边缘计算协作式资源分配策略[J]. 电讯技术,2021,61(7):858-864.

[2] SADI F, FILEGGI L, FRANCHETTI F. Algorithm and hardware cooptimized solution for large SPMV problems [C]//Proceedings of 2017 IEEE High Performance Extreme Computing Conference. Waltham: IEEE, 2017:1-7.

[3] 王晞阳,陈继林,李猛,等. FPGA 架构上面向稀疏矩

阵求解的静态调度算法[J]. 计算机工程,2022,48(7):199-205.
[4] 吴志勇,王晞阳,陈继林. 一种基于 FPGA 并行加速的稀疏矩阵求解方法[J]. 电力系统保护与控制,2021,49(11):155-162.
[5] HEGDE K, ASGHARI H, PELLAURE M, et al. ExTensor:an accelerator for sparse tensor algebra[C]//Proceedings of the 52nd Annual IEEE/ACM International Symposium. Columbus:ACM,2019:1-7.
[6] SRIVASTAVA N,JIN H,SMITH S,et al. Tensaurus:a versatile accelerator for mixed sparse-dense tensor computations [C]//Proceedings of 2020 26th International Symposium on High-Performance Computer Architecture. California:IEEE,2020:1-10.
[7] SRIVASTAVA N, JIN H, LIU J, et al. MatRaptor: a sparse-sparse matrix multiplication accelerator based on row-wise product[C]//Proceedings of 2020 53rd IEEE/ACM International Symposium on Microarchitecture. Athens:ACM,2020:1-11.
[8] 谢震,谭光明,孙凝晖. 基于 PPR 模型的稀疏矩阵向量乘及卷积性能优化研究[J]. 计算机研究与发展,2021,58(3):445-457.
[9] 李亿渊,薛巍,陈德训,等. 稀疏矩阵向量乘法在神威众核架构上的性能优化[J]. 计算机学报,2020,43(6):1010-1024.
[10] 刘珊瑕,靳松,王文琛,等. 基于变量相关性分解方法的稀疏线性方程组并行求解算法[J]. 清华大学学报(自然科学版),2020,60(4):312-320.
[11] 杨飞,马昱春,侯金,等. 基于 MPSoC 并行调度的矩阵乘法加速算法研究[J]. 计算机科学,2017,44(8):36-41.
[12] DORRANCE R,REN F,MARKOVI D. A scalable sparse matrix-vector multiplication kernel for energy-efficient sparse-blas on FPGAs[C]//Proceedings of 2014 ACM/SIGDA International Symposium on Field-programmable Gate Arrays. California:ACM,2014:161-170.
[13] ASGARI B,HADIDI R,KIM H. ASCELLA:accelerating sparse computation by enabling stream accesses to memory[C]//Proceedings of 2020 Design,Automation& Test in Europe Conference & Exhibition. Grenoble:IEEE,2020:1-4.
[14] 顾越,赵银亮. 基于 RISC-V 向量指令的稀疏矩阵向量乘法实现与优化[J]. 计算机工程与科学,2022,44(1):1-8.
[15] Xilinx Semiconductor. ZCU102 evaluation board user guide[EB/OL]. (2019-06-12)[2022-09-10]. http://www.xilinx.com.

作者简介:

朱明达　男,1983 年生于山东东营,2014 年获博士学位,现为副教授,主要研究方向为电子测量技术与仪器、信号检测与处理、智能信息处理。

薛济擎　女,1997 年生于辽宁抚顺,2019 年获学士学位,现为硕士研究生,主要研究方向为智能信息处理。

艾纯瑶　女,1997 年生于辽宁丹东,2019 年获学士学位,现为硕士研究生,主要研究方向为智能信息处理。