计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2019.20170908
56(2):410-420, 2019

双精度浮点矩阵乘协处理器研究

贾 迅 邬贵明 谢向辉 吴 东
(数学工程与先进计算国家重点实验室 江苏无锡 214125)
(jia.xun@meac-skl.cn)

A Coprocessor for Double-Precision Floating-Point Matrix Multiplication

Jia Xun, Wu Guiming, Xie Xianghui, and Wu Dong
(*State Key Laboratory of Mathematical Engineering and Advanced Computing*, *Wuxi*, *Jiangsu* 214125)

Abstract Matrix multiplication has been widely used in various application fields, especially the field of numerical computation. However, double-precision floating-point matrix multiplication suffers from non-optimal performance or efficiency on contemporary computing platforms, including CPU, GPGPU and FPGA. To address this problem, acceleration of double-precision floating-point matrix multiplication with a customized coprocessor is proposed in this paper, which adopts linear array as the basic building block. Firstly, double-buffering technique and optimized memory scheduling are applied to the basic linear array for better computation efficiency. Then, architecture of the matrix multiplication coprocessor and coprocessor-based accelerated computing system are formulated. Furthermore, a performance model tailored for the coprocessor is developed and the design space of coprocessor is explored in detail. Finally, functional correctness of the coprocessor is verified and its hardware implementation cost under mainstream technology node is evaluated. Experimental results show that the proposed coprocessor can achieve the performance of 3 TFLOPS and the efficiency of 99%. Compared with NVIDIA K40 GPGPU for executing double-precision floating-point matrix multiplication, the coprocessor proposed in this paper achieves 1.95× performance with hardware overheads of only 21.05% in area. This work explores the application of customized acceleration in high-performance computing and has certain guidance for improving performance of existing computing systems.

Key words matrix multiplication; coprocessor; acceleration; floating-point; hardware customization

摘 要 矩阵乘运算在多个应用领域特别是数值计算领域被广泛使用,但双精度浮点矩阵乘在CPU,GPGPU,FPGA等现有计算平台上的性能和效率受限,其往往成为大规模数值计算应用的性能瓶颈.针对该问题,以线性阵列计算结构为基础,研究了双精度浮点矩阵乘的定制加速.首先,对线性阵列计算结构进行了双缓冲优化并设计了针对双缓冲的存储访问调度,以提高结构的计算效率.其次,提出了矩阵乘协处理器和加速计算系统的结构,构建了协处理器的性能模型并对其结构设计空间进行了探索.最后,验证了协处理器的功能正确性并在某主流工艺下评估了其硬件开销.实验结果表明,设计的双精度浮点矩阵乘协处理器可以达到3 TFLOPS的计算性能和99%的计算效率.与NVIDIA K40 GPGPU相比,协处理器执行双精度浮点矩阵乘的性能是K40的1.95倍,而面积开销仅为K40的21.05%.探索了定制加速结构设计在高性能计算中的应用,对现有计算系统的性能提升具有一定的参考价值.

收稿日期:2017-11-28;**修回日期**:2018-03-05
基金项目:国家自然科学基金项目(91430214,61732018)
This work was supported by the National Natural Science Foundation of China (91430214, 61732018).

关键词　矩阵乘;协处理器;加速;浮点;硬件定制

中图法分类号　TP302

矩阵乘运算核心广泛应用于数值计算、数字信号分析、图像处理、人工智能等领域,其计算形式为

$$\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\sum_{k=0}^{K-1}C[i,j]+=A[i,k]\times B[k,j],$$

其中,$\boldsymbol{A}\in\mathbb{R}^{M\times K}$,$\boldsymbol{B}\in\mathbb{R}^{K\times N}$,$\boldsymbol{C}\in\mathbb{R}^{M\times N}$.循环 k 的计算存在写后读数据相关,无法并行;循环 i 和循环 j 的计算不存在真相关,可以并行.另外,计算过程中的存储访问均比较规整.由于计算复杂度为 $O(n^3)$,双精度浮点矩阵乘往往是大规模数值计算应用的性能瓶颈[1],其性能直接反映了系统的计算能力.以性能测试程序 Linpack[2] 为例,程序实现了基于双精度浮点矩阵乘的线性方程求解,并以单位时间内系统执行浮点操作的次数(FLOPS)来衡量系统的计算能力.因此,为双精度浮点矩阵乘运算提供高性能且高效的计算平台对大规模数值计算应用的性能发挥至关重要.

目前,矩阵乘的计算平台主要包括 CPU,GPGPU,FPGA.其中,CPU 和 GPGPU 平台上矩阵乘的实现和应用较为成熟.处理器厂商会提供针对各自产品架构进行性能优化的计算函数接口库,如 Intel MKL[3],IBM ESSL[4],NVIDIA cuBLAS[5] 等.由于处理器架构面向通用计算,CPU 和 GPGPU 上矩阵乘的计算效率无法达到最优,如 Intel Knights Corner 众核处理器矩阵乘的计算效率仅为 89%[6],NVIDIA P100 的计算效率约为 93%[7].矩阵乘效率对系统计算效率的影响几乎是线性的,矩阵乘效率的损失必然导致系统计算效率的降低[8].

FPGA 计算平台具有灵活的可编程性、细粒度的并行能力和丰富的逻辑资源.同时,矩阵乘运算数据并行性好和存储访问规整的算法特点适合并行计算结构的设计[9],因此基于 FPGA 平台实现针对矩阵乘的并行计算结构成为学术研究的热点,并且已经取得一些研究成果[10].以浮点乘加运算部件作为计算单元(processing elements, PE)的核心,多个计算单元组成广播[11]或阵列结构[12-14]进行加速计算是典型的实现方式.其中文献[12]提出的线性阵列计算结构可以处理任意规模的矩阵乘运算,且数据传输和存储的效率较高.由于矩阵乘的性能受限于 FPGA 的可用的逻辑资源和时钟频率,即使以资源利用和时钟频率为优化目标的结构设计,其计算性能也只能达到 203.1 GFLOPS[15](252 个 PE,时钟频率为 403 MHz).显然,目前基于 FPGA 的矩阵乘无法满足大规模数值计算应用对计算能力的需求.

近年来,面向特定应用的硬件定制加速成为高性能、高效计算结构设计的一个重要趋势[16-17].Google 公开了 TPU(tensor processing unit)芯片的设计[18],其核心是整数矩阵乘脉动阵列计算结构.NVIDIA 最新发布的 V100 GPGPU[19] 芯片中引入了称为"Tensor Core"的 16 b,32 b 混合精度矩阵乘阵列计算结构,单芯片集成了 672 个 TensorCore,可提供 120 TFLOPS 的计算能力.由于 TPU 和 V100 芯片中矩阵乘定制加速结构的设计面向深度学习类应用,结构支持的计算精度较低[20-21],因而无法直接应用于对计算精度要求较高的数值计算领域.

本文以线性阵列计算结构为基础,对其进行结构优化,并采用硬件定制的方法设计实现了支持双精度浮点运算的矩阵乘协处理器.同时,本文建立了性能模型并深入分析了各结构设计参数对协处理器实际计算性能和效率的影响.此外,本文还验证了矩阵乘协处理器的功能正确性并评估了其硬件实现的开销.本文探索了硬件定制结构设计在双精度浮点矩阵乘加速计算中的应用,研究成果对提升现有计算系统的性能和效率有一定的借鉴意义.

1　线性阵列计算结构及大规模矩阵乘算法

线性阵列计算结构[12]如图 1 所示,多个计算单元线性互连,每个计算单元包含局部存储器 c 以存储矩阵 $\boldsymbol{C}$ 的分块数据,寄存器 a 和 b 分别存储矩阵 $\boldsymbol{A}$ 和矩阵 $\boldsymbol{B}$ 的数据元素,一个双精度浮点乘加运算部件和一个计数器 CNT.

数据 b 在 PE 之间进行传输,其有效时驱动 PE 执行一次乘加计算:$c[CNT]=c[CNT]+a\times b$.

算法 1 给出了基于上述线性阵列计算结构的大规模分块矩阵乘算法[12].参数 S_p,S_t 分别是矩阵 $\boldsymbol{C}$ 行数 M 和列数 N 的分块粒度,同时也表示线性阵列计算结构中 PE 的数目和各 PE 集成的局部存储器 c 的深度.算法中索引为 p 和 t 的最内层循环对应线性阵列计算结构执行的浮点矩乘迭代计算,即 S_p 个 PE 实现矩阵 $\boldsymbol{A}$ 中规模为 $S_p\times 1$ 的矩阵子块与矩阵 $\boldsymbol{B}$ 中规模为 $1\times S_t$ 的矩阵子块乘计算,并将计算结果与矩阵 $\boldsymbol{C}$ 中规模为 $S_p\times S_t$ 的子块相加.

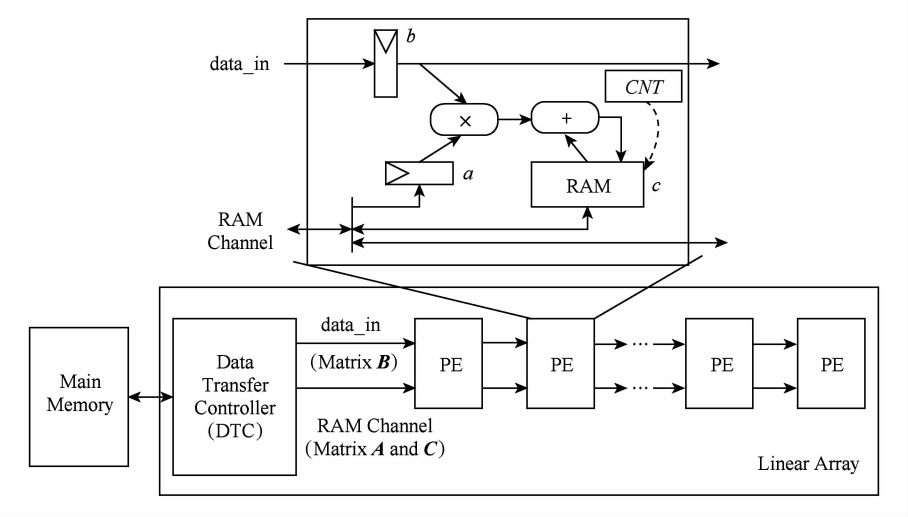


Fig. 1 Linear array computing architecture

图 1 线性阵列计算结构

阵列计算结构在单次迭代计算中共执行 $2S_pS_t$ 次双精度浮点运算.

算法 1. 基于线性阵列结构的大规模矩阵乘算法.

输入:双精度浮点矩阵 $\boldsymbol{A}$、矩阵 $\boldsymbol{B}$、矩阵 $\boldsymbol{C}$;

输出:双精度浮点矩阵 $\boldsymbol{C}$.

① For $T_p=0$ to $M-1, S_p$

② For $T_t=0$ to $N-1, S_t$

③ Load $C[T_p:T_p+S_p-1, T_t:T_t+S_t-1]$;

④ For $k=0$ to $K-1$

⑤ Load $A[T_p:T_p+S_p-1, k]$;

⑥ Load $B[k, T_t:T_t+S_t-1]$;

⑦ For $p=T_p$ to T_p+S_p-1

⑧ For $t=T_t$ to T_t+S_t-1

⑨ $C[p,t]+=A[p,k]\times B[k,t]$;

⑩ End For

⑪ End For

⑫ End For

⑬ Store $C[T_p:T_p+S_p-1, T_t:T_t+S_t-1]$;

⑭ End For

⑮ End For

算法 1 中索引为 T_p, T_t, k 的 3 层循环在线性阵列计算结构进行矩阵乘迭代计算的同时,处理矩阵分块数据的加载与写回,其功能在硬件上由数据传输控制部件 DTC(data transfer controller)实现. 各矩阵分块数据到线性阵列计算结构的映射如图 2 所示:

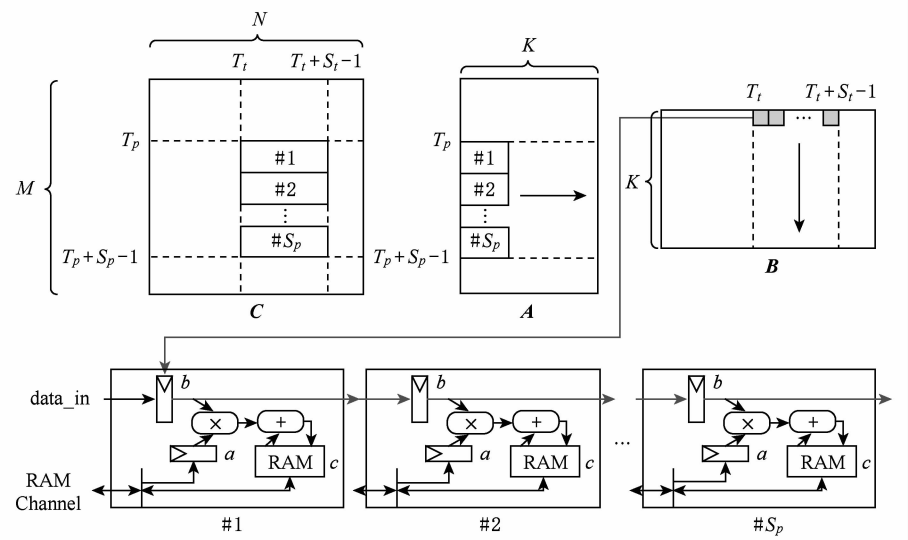


Fig. 2 Data block mapping for the linear array

图 2 矩阵分块数据到阵列计算结构的映射

线性阵列结构经过 K 轮迭代计算才能得到矩阵 $\boldsymbol{C}$ 中一个分块数据的计算结果.因此,在 K 轮迭代计算开始前,数据传输控制部件需要先将矩阵 $\boldsymbol{C}$ 中规模为 $S_p \times S_t$ 的分块数据 $C[T_p:T_p+S_p-1, T_t:T_t+S_t-1]$ 分别加载到 S_p 个计算单元的深度为 S_t 的局部存储器 c 中,并在 K 轮迭代计算完成后将对应的分块计算结果写回主存.另外,每次迭代计算中各 PE 需要访问矩阵 $\boldsymbol{A}$ 中相同的数据元素.因此,每次迭代计算开始前,数据传输控制部件还需要将矩阵 $\boldsymbol{A}$ 中规模为 $S_p \times 1$ 的分块数据 $A[T_p:T_p+S_p-1,k]$ 加载到 S_p 个计算单元的寄存器 a 中.

矩阵 $\boldsymbol{A}$ 和矩阵 $\boldsymbol{C}$ 的分块数据全部加载完成后,数据传输控制部件再加载矩阵 $\boldsymbol{B}$ 的数据元素以驱动阵列计算结构进行迭代计算.矩阵乘运算执行完成共需 $K\times(M/S_p)\times(N/S_t)$ 次迭代计算.对计算过程中数据传输控制部件传输矩阵 $\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}$ 分块数据的数据量、时间间隔以及次数的量化分析如表 1 所示:

Table 1　Analysis on Data Transfer for Matrix Multiplication

表 1　针对矩阵乘运算过程中数据传输的分析

Matrix	Amount of Data	Interval	Number of Transfer
$\boldsymbol{A}$	S_p	S_t	$K\times\frac{M}{S_p}\times\frac{N}{S_t}$
$\boldsymbol{B}$	1	1	$K\times S_t\times\frac{M}{S_p}\times\frac{N}{S_t}$
$\boldsymbol{C}$	$S_p\times S_t$	$K\times S_t$	$2\times\frac{M}{S_p}\times\frac{N}{S_t}$

文献[12]基于 Xilinx 公司 Virtex-5 系列的 FPGA 实现了上述线性阵列计算结构.PE 个数 S_p 设置为 96、局部存储器深度 S_t 设置为 1 024 的情况下,FPGA 的最大运行频率为 158.08 MHz.输入矩阵的规模 M,N,K 均为 8 192 时,阵列计算结构完成双精度浮点矩阵乘运算所需的时间为 40.48 s,即结构的计算性能为 27.16 GFLOPS,效率为 89.49%.

线性阵列计算结构虽然存储效率较高,但每次迭代计算中各 PE 需要等待计算所需的矩阵分块数据全部加载完成后才能开始计算,计算与存储访问在时间上是串行的,计算资源的闲置必然导致计算效率的损失,高效计算结构的设计必须对此进行改进.由于结构设计面向 FPGA 计算平台,文献[12]仅以 FPGA 可用的逻辑资源作为设计约束,并未考虑局部存储器的深度、输入矩阵的规模等参数对线性阵列计算结构实际计算性能和效率的影响,而这些参数在定制协处理器的设计中尤为重要.另外,本文协处理器的设计除了考虑核心的计算结构外,还需要考虑其与主机端的接口,以实现控制信息和计算数据的传输,从而构成完整的加速计算系统.

2　协处理器的设计

2.1　线性阵列计算结构的优化

根据算法 1 给出的大规模分块矩阵乘运算算法,线性阵列计算结构在进行矩阵乘迭代计算前,需要先从主存读入矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{C}$ 的分块数据;K 轮迭代计算完成后,需要先将矩阵 $\boldsymbol{C}$ 的分块计算结果写回主存,才能开始下一轮的迭代计算.矩阵分块数据在主存和线性阵列计算结构之间进行传输的过程中,结构中各计算单元无法进行计算,计算资源被闲置,从而导致线性阵列计算结构整体的效率较低.针对这个问题,本文采用双缓冲技术对阵列计算结构中各计算单元的设计进行了优化.优化后,计算单元的结构如图 3 所示:

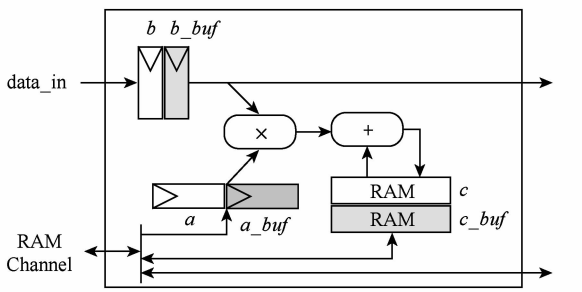


Fig. 3　Structure of PE optimized with double buffering

图 3　采用双缓冲优化技术的 PE 结构

图 3 所示的 PE 结构中除了已有的寄存器 a,b 和局部存储器 c 之外,还包括双缓冲优化引入的存储矩阵 $\boldsymbol{A},\boldsymbol{B}$ 数据元素的寄存器 a_buf,b_buf 以及存储矩阵 $\boldsymbol{C}$ 分块数据的局部存储器 c_buf.

应用双缓冲优化技术后,阵列计算结构可以在使用当前缓冲中的数据进行迭代计算的同时,提前将下一轮迭代计算所需的矩阵分块数据装入到另一缓冲,实现计算与访存的重叠,隐藏存储访问的开销.此时,高效的存储访问调度成为应用双缓冲优化提升计算结构资源利用率的关键.

2.2　存储访问调度

由表 1 列出的矩阵乘运算过程中矩阵分块数据的传输可知,线性阵列计算结构在进行矩阵乘运算时,每个时钟周期仅需访问矩阵 $\boldsymbol{B}$ 的一个数据元素,其对存储带宽的需求较低;而矩阵 $\boldsymbol{A},\boldsymbol{C}$ 的访问时间间隔分别为 S_t 和 $K\times S_t$ 个时钟周期,因此可以利用读取矩阵 $\boldsymbol{B}$ 数据元素剩余的存储访问带宽

提前加载矩阵 $\boldsymbol{A}$,$\boldsymbol{C}$ 的分块数据至 PE 的缓冲中,或将矩阵 $\boldsymbol{C}$ 的分块计算结果从缓冲写回主存.

算法 2 给出了基于双缓冲的存储访问调度算法.数据传输控制部件在硬件上实现了该算法,并支持双缓冲读写访问的控制.

算法 2. 基于双缓冲的存储访问调度算法.

输入:双精度浮点矩阵 $\boldsymbol{A}$、矩阵 $\boldsymbol{B}$、矩阵 $\boldsymbol{C}$;

输出:双精度浮点矩阵 $\boldsymbol{C}$.

① Load $C[0:S_p-1,0:S_t-1]$;
② Load $A[0:S_p-1,0]$;
③ For $T_p=0$ to $M-1,S_p$
④ For $T_t=0$ to $N-1,S_t$
⑤ For $k=0$ to $K-1$
⑥ Load $B[k,T_t:T_t+S_t-1]$;
Load $A[T_p:T_p+S_p-1,k+1]$ or
Store $C[T_p:T_p+S_p-1,T_t-S_t:T_t-1]$ or
Load $C[T_p:T_p+S_p-1,T_t+S_t-1:T_t+2S_t-1]$;
⑦ For $p=T_p$ to T_p+S_p-1
⑧ For $t=T_t$ to T_t+S_t-1
⑨ $C[p,t]+=A[p,k]\times B[k,t]$;
⑩ End For
⑪ End For
⑫ End For
⑬ End For
⑭ End For
⑮ Store $C[M-S_p:M-1,N-T_p:N-1]$.

算法 2 中,阵列计算结构进行第一次迭代计算所需的矩阵 $\boldsymbol{A}$,$\boldsymbol{C}$ 的分块数据先被读入.读取矩阵 $\boldsymbol{B}$ 的数据元素进行迭代计算时,优先利用剩余的存储访问带宽提前读取下一次迭代计算所需的矩阵 $\boldsymbol{A}$ 的分块数据;待矩阵 $\boldsymbol{A}$ 的分块数据读入完成,再利用剩余带宽写回前 K 轮迭代计算得到的矩阵 $\boldsymbol{C}$ 的分块计算结果;待计算结果写回完成,最后利用剩余带宽装入后 K 轮迭代计算所需的矩阵 $\boldsymbol{C}$ 的分块数据.所有迭代计算完成后,矩阵 $\boldsymbol{C}$ 中最后一个分块计算结果被写回.

理想情况下,线性阵列计算结构在开始新一次的迭代计算时,各 PE 所需的矩阵数据均已在缓冲中,即存储访问的延迟可以有效地被计算隐藏,结构中的计算资源可以得到充分利用.而实际的存储访问调度受限于存储带宽、计算单元数、局部存储器的深度以及矩阵规模等参数,其对矩阵乘协处理器实际性能的影响将在 2.3 节进行分析.

2.3 系统结构

基于矩阵乘协处理器的加速计算系统的整体结构如图 4 所示.系统由主机端处理器芯片和双精度浮点矩阵乘协处理器芯片 2 部分组成.其中,矩阵乘协处理器通过 I/O 总线与主机端处理器相连.系统采用 PCIe-3.0×16 作为芯片间的互连接口,双向通信带宽最高为 31.5 GBps.

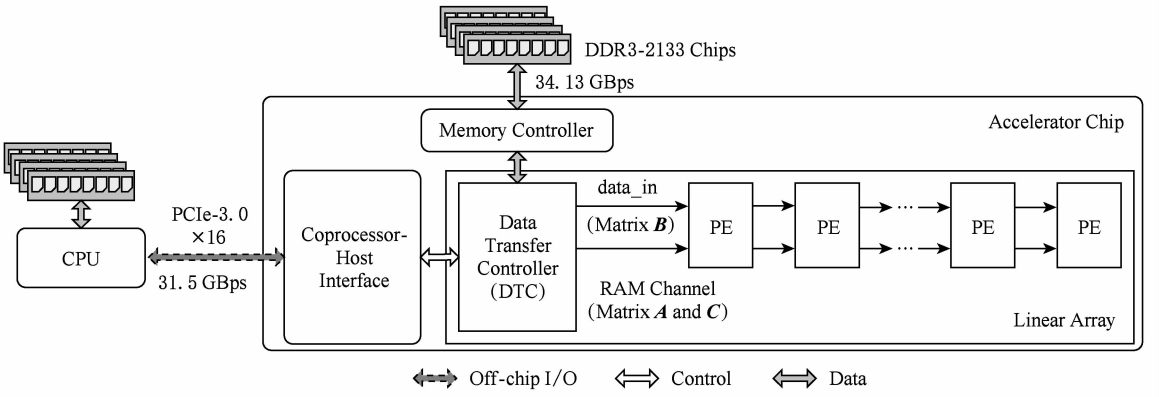


Fig. 4 System architecture based on matrix multiplication coprocessor

图 4 基于矩阵乘协处理器计算系统的架构

主机端和协处理器端存储空间相互独立,待计算的矩阵数据存储在协处理器端的内存.本文协处理器的设计面向大规模矩阵乘,支持的矩阵规模至少为 16 384×16 384.因此,主机端与协处理器端的内存容量配置为 8 GB,并采用 DDR3-2133[22] 规格的存储芯片,接口的数据宽度为 128 b,带宽为 34.13 GBps.

系统对主机端处理器的类型没有限制,只要支持 PCIe 传输协议即可,其指令集系统架构可以是 x86,Power,ARM,RISC-V[23] 等.矩阵乘协处理器由应用双缓冲优化的线性阵列计算结构、存储访问控制器、协处理器与主机端接口(coprocessor-host interface, CHI)3 部分构成.乘加运算部件是计算

单元的核心,本文采用全流水且兼容 IEEE-754 浮点数标准[24]的乘加运算部件设计,以保证协处理器的计算性能. CHI 接口实现主机端与协处理器之间控制和数据的传输. 主机端向矩阵乘协处理器发送的主要命令如下:

1) *Initialization* 对协处理器的控制与状态寄存器进行初始化;

2) *DataIn* 将待计算矩阵数据从主机端内存加载到协处理器端内存;

3) *MatrixMultiply* 启动协处理器上的双精度浮点矩阵乘运算;

4) *DataOut* 将矩阵乘运算的结果从协处理器端内存读出到主机端内存.

接收到来自主机端发送的命令,CHI 接口对命令进行分析并生成线性阵列计算结构相关的控制信号. 协处理器在进行矩阵乘运算的过程中,主机端与协处理器之间不需要进行数据传输,从而简化了协处理器的设计与调试.

应用程序在计算系统上加速矩阵乘运算的过程如图 5 所示. 当矩阵乘协处理器连接至主机端时,主机端发送 *Initialization* 命令对协处理器进行初始化,如设置浮点计算的舍入模式、溢出控制等. 应用程序运行在主机端,当其调用双精度浮点矩阵乘运算核心,如基础线性代数子程序库(basic linear algebra subroutines, BLAS)中的函数 *blas_dgemm* $(M,N,K,\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\cdots)$时,主机端通过 CHI 接口向协处理器发送 *DataIn* 命令和函数调用传入的矩阵规模、矩阵数据在内存中的起始地址等参数. 协处理器的数据传输控制部件将待计算的矩阵数据从主机端内存以直接内存访问(direct memory access, DMA)方式传输至协处理器端内存. 数据传输完成后,主机端发送 *MatrixMultiply* 命令以启动协处理器上的矩阵乘运算. 运算过程中,数据传输控制部件按算法 2 设计的存储访问调度实现矩阵分块数据在协处理器端内存和线性阵列计算结构之间的传输.

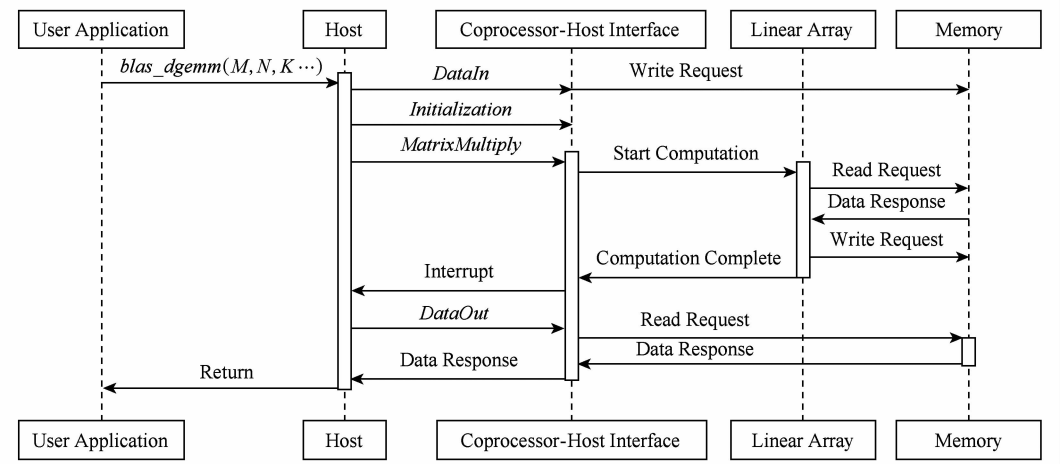


Fig. 5　Computation based on matrix multiplication coprocessor

图 5　基于矩阵乘协处理器的计算过程

协处理器完成矩阵乘运算后,CHI 接口向主机端发出中断信号. 主机端收到中断后,向 CHI 接口发送 *DataOut* 命令,协处理器的数据传输控制部件以 DMA 的方式将矩阵乘运算的结果从协处理器端内存传输回主机端内存. 运算结果传输完成后,矩阵乘运算调用结束,应用程序继续执行.

3　性能模型

为了量化分析双精度浮点矩阵乘协处理器的各结构设计参数对其实际计算性能和效率的影响,本文基于算法 2 设计的存储访问调度算法构建了协处理器的性能模型. 模型中的基本参数包括线性阵列计算结构中 PE 的个数 S_p、PE 集成的局部存储器的深度 S_l 以及输入矩阵的规模 M,N,K. 此外,本文将模型中协处理器的时钟频率记为 $Freq$(单位为 GHz),协处理器端内存的访问带宽记为 $Band$(单位为 GBps),矩阵乘运算核心在协处理器上执行完成所需的时间记为 T(单位为 s),协处理器的实际计算性能和效率记为 $Perf$(单位为 GFLOPS)和 Eff.

当计算过程中的存储访问延迟完全被计算隐藏,协处理器的计算性能可以充分发挥,此时的计算效率最高. 就本文提出的存储访问调度算法,每次迭代计算提前读入矩阵 $\boldsymbol{A}$ 中规模为 S_p 的分块数据的

延迟与读入矩阵 $\boldsymbol{B}$ 中规模为 S_t 的分块数据的延迟之和应当小于等于对应的计算延迟,即 S_t 个时钟周期.同理,K 轮迭代计算的总存储访问延迟应当小于等于对应的总计算延迟,即 $K\times S_t$ 个时钟周期.存储访问的数据粒度为 8 B 的双精度浮点数,上述 2 个约束条件可表示为

$$\begin{cases}8\times\dfrac{S_p+S_t}{Band}\leqslant\dfrac{S_t}{Freq}\\ 8\times\dfrac{K\times(S_p+S_t)+2\times S_p\times S_t}{Band}\leqslant\dfrac{K\times S_t}{Freq}\end{cases}\Rightarrow$$

$$\frac{S_p}{S_t}+\frac{2S_p}{K}\leqslant\frac{Band}{8\times Freq}-1. \tag{1}$$

协处理器端 DDR3-2133 存储芯片的数据传输率为 2.133 GT/s,单次传输的数据宽度为 128 b.若协处理器的工作时钟频率 $Freq=1.5$ GHz,则其每个时钟周期可访问 2 个双精度浮点数.此时,式(1)的约束条件可进一步改写为

$$\frac{S_p}{S_t}+\frac{2S_p}{K}\leqslant 1. \tag{2}$$

若参数 S_p,S_t,K 可以满足式(2)中的约束条件,则计算过程中的存储访问延迟可以完全被计算隐藏.此时,矩阵乘运算核心在协处理器上的执行完成所需的时间 T 最小.其由 3 部分时间构成,包括读入矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{C}$ 第 1 个子块数据的时间、迭代计算的时间以及将矩阵 $\boldsymbol{C}$ 中最后 1 个子块的计算结果写回内存的时间.计算时间 T 可表示为

$$T=\frac{S_pS_t+S_p}{2}+\frac{M}{S_p}\times\frac{N}{S_t}\times KS_t+\frac{S_pS_t}{2}. \tag{3}$$

若参数 S_p,S_t,K 不满足式(2)的约束条件,存在 $S_t<S_p$ 和 $S_p<S_t$,$2S_p/K>1-S_p/S_t$ 这 2 种可能.下文将分别分析这 2 种情况下矩阵乘运算核心在协处理器上的执行时间.

对于第 1 种情况,提前读入矩阵 $\boldsymbol{A}$ 子块数据的存储访问延迟无法完全被计算时间隐藏,每次迭代计算的时间增加 $(S_p-S_t)/2$ 个时钟周期.根据算法 2,当前迭代计算执行过程中,写回前 K 轮迭代计算中矩阵 $\boldsymbol{C}$ 子块计算结果和提前读入后 K 轮迭代计算中矩阵 $\boldsymbol{C}$ 子块计算数据的存储访问延迟均无法被计算隐藏,每 K 轮迭代计算的时间增加 $S_p\times S_t$ 个时钟周期.此时,矩阵乘运算核心在协处理器上执行完成所需的时间 T 可表示为

$$T=\frac{S_pS_t+S_p}{2}+\frac{M}{S_p}\times\frac{N}{S_t}\times\left(K\left(S_t+\frac{S_p-S_t}{2}\right)+S_pS_t\right). \tag{4}$$

对于第 2 种情况,当前迭代计算的执行时间可以完全隐藏提前读入矩阵 $\boldsymbol{A}$ 子块数据的存储访问延迟,但只能部分隐藏写回前 K 轮迭代计算中矩阵 $\boldsymbol{C}$ 子块计算结果和提前读入后 K 轮迭代计算中矩阵 $\boldsymbol{C}$ 子块计算数据的存储访问延迟,其导致每 K 轮迭代计算的时间增加 $S_p\times S_t-K\times(S_t-S_p)/2$ 个时钟周期.此时,矩阵乘运算核心在协处理器上执行完成所需的时间 T 可表示为

$$T=\frac{S_pS_t+S_p}{2}+\frac{M}{S_p}\times\frac{N}{S_t}\times\left(KS_t+S_pS_t-\frac{K}{2}(S_t-S_p)\right). \tag{5}$$

协处理器完成矩阵乘核心的加速计算共需执行 $2MNK$ 次双精度浮点操作.基于对 3 种情况下矩阵乘运算核心在协处理器上执行时间 T 的分析结果,可以构建式(6)所示的协处理器的性能模型.此时,协处理器的计算效率可表示为实际计算性能与峰值计算性能的比值,即 $Eff=Perf/(2\times S_p\times Freq)$.

$$Perf=\begin{cases}\dfrac{2MNK\times Freq}{\dfrac{S_pS_t+S_p}{2}+\dfrac{M}{S_p}\times\dfrac{N}{S_t}\times KS_t+\dfrac{S_pS_t}{2}}, & \dfrac{S_p}{S_t}+\dfrac{2S_p}{K}\leqslant 1;\\ \dfrac{2MNK\times Freq}{\dfrac{S_pS_t+S_p}{2}+\dfrac{M}{S_p}\times\dfrac{N}{S_t}\times\left(K\left(S_t+\dfrac{S_p-S_t}{2}\right)+S_pS_t\right)}, & S_t<S_p;\\ \dfrac{2MNK\times Freq}{\dfrac{S_pS_t+S_p}{2}+\dfrac{M}{S_p}\times\dfrac{N}{S_t}\times\left(KS_t+S_pS_t-\dfrac{K}{2}(S_t-S_p)\right)}, & S_t>S_p\ 且\ \dfrac{2S_p}{K}>1-\dfrac{S_p}{S_t}.\end{cases} \tag{6}$$

4 实验与结果

本节以第 3 节提出的性能模型为基础,分析矩阵乘协处理器结构设计中双缓冲优化技术、局部存储器的深度和输入矩阵规模等参数对协处理器实际计算性能和效率的影响.实验假设线性阵列计算结构的 PE 数为 1024,协处理器的时钟频率为 1.5 GHz,待计算的矩阵均为方阵,即 $M=N=K=n$.此外,本节还验证了矩阵乘协处理器功能实现的正确性;

在主流工艺下评估了其硬件实现的开销,并与同等工艺的 GPGPU 芯片进行了对比,以探索其硬件可实现性和在实际应用中的优势.

4.1　双缓冲优化技术对矩阵乘协处理器性能的影响

实验在不同的矩阵规模下,分别统计了应用双缓冲优化技术前后矩阵乘协处理器的性能,以分析应用双缓冲优化技术对协处理器性能的影响.实验中,各 PE 局部存储器的深度均设置为 2 048,实验的结果如图 6 所示.

从图 6 可以看出,不同矩阵规模下,双缓冲优化技术的应用均可以提升矩阵乘协处理器的计算性能,协处理器的性能平均提升 30.06%.矩阵规模为 4 096 时,性能提升最为明显,达到了 45.45%.可见,本文设计的存储访问调度算法可以在计算过程中利用双缓冲优化有效隐藏存储访问的延迟.

同时,双缓冲优化对矩阵乘协处理器的性能提升受矩阵规模的影响.对此分析如下:当矩阵规模较小时,计算无法有效隐藏存储访问的延迟,应用双缓冲优化对性能的提升作用有限;随着矩阵规模逐渐增加,计算可以隐藏更多的访存延迟,此时应用双缓冲优化可以显著提升协处理器的计算性能;当矩阵规模较大时,计算可以很好隐藏存储访问的延迟,而实验中局部存储器的深度保持不变,双缓冲优化对性能的提升作用逐渐降低.

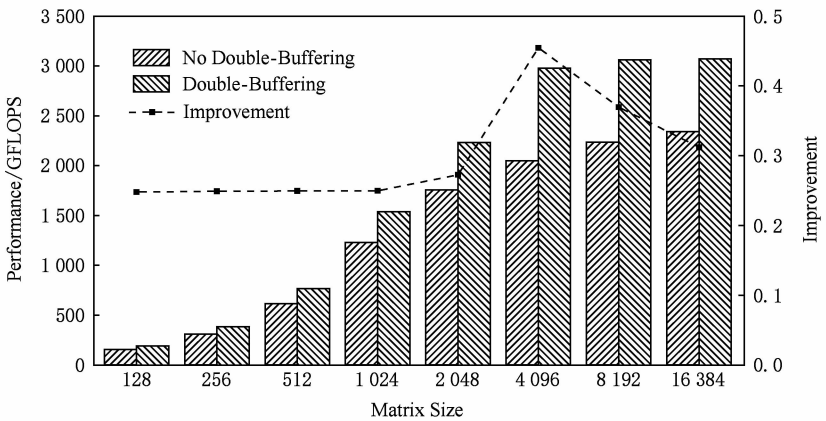


Fig. 6　Effect of double-buffering optimization technique on coprocessor performance

图 6　双缓冲优化技术对矩阵乘协处理器性能的提升

4.2　局部存储器深度对矩阵乘协处理器性能的影响

局部存储器的深度直接影响协处理器硬件实现的开销,因而是矩阵乘协处理器结构设计中一个重要的参数.实验在 128,1 024,8 192,16 384 这 4 种矩阵规模下,分别将计算单元局部存储器的深度 h 从 128 增加至 8 192,统计矩阵乘协处理器实际计算性能的变化情况,实验结果如图 7 所示:

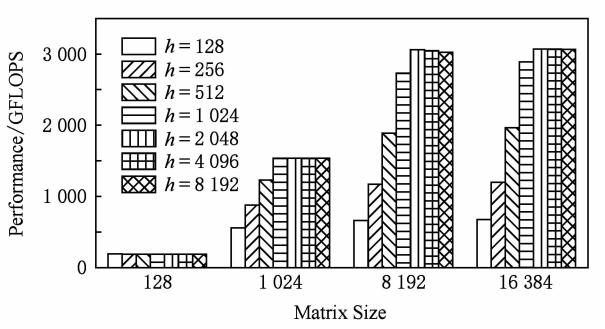


Fig. 7　Effect of local memory depth on coprocessor performance

图 7　局部存储器深度对矩阵乘协处理器性能的影响

从图 7 可以看出,4 种矩阵规模下,矩阵乘协处理器的实际计算性能均随局部存储器深度的增加而提升,局部存储器深度增加大于 2 048 时,矩阵乘协处理器的性能基本保持不变.协处理器实际计算性能随局部存储器深度的变化可以根据式(2)中计算完全隐藏存储访问延迟的约束条件进行分析.当 $S_t < S_p$ 时,提前加载矩阵 $\boldsymbol{A}$ 中子块数据的延迟无法完全被计算隐藏,此时增加局部存储器的深度可以有效提升协处理器的计算性能.而当局部存储器的深度大于 PE 个数时,协处理器的实际计算性能不再受局部存储器深度的影响,因而其保持不变.基于上述实验结果和分析,当协处理器线性阵列计算结构的 PE 数为 1 024 时,局部存储器最优的深度为 2 048.

4.3　矩阵规模对矩阵乘协处理器计算效率的影响

尽管本文矩阵乘协处理器的设计面向数值计算领域的大规模应用,矩阵规模对协处理器实际计算性能和效率的影响仍然值得关注.实验在应用双缓冲优化、局部存储器深度设置为 2 048 时,统计了不同矩阵规模下,矩阵乘协处理器实际的计算性能和

效率,以及存储访问延迟占矩阵乘运算核心总执行时间的比例,实验结果如表 2 所示:

Table 2 Effect of Matrix Size on Coprocessor Performance and Efficiency

表 2 矩阵规模对矩阵乘协处理器性能和效率的影响

n	Memory Ratio/%	Performance	Efficiency/%
128	50.00	191.97	6.25
256	50.00	383.98	12.50
512	50.00	767.99	25.00
1024	27.27	1535.99	50.00
2048	3.03	2234.28	72.73
4096	0.04	2978.93	96.97
8192	0.00	3060.04	99.61
16384	0.00	3070.50	99.97

从表 2 中的数据可以看出,随着矩阵规模的增加,矩阵乘协处理器的实际计算效率不断提高:矩阵规模为 128 时,协处理器的计算效率仅为 6.25%;矩阵规模增加至 4096 时,协处理器的计算效率达到 96.97%;矩阵规模大于等于 8192 时,协处理器的计算效率可以达到 99%.

对协处理器计算效率与矩阵规模相关性的分析如下:当 PE 数 $S_p=1024$、局部存储器深度 $S_t=2048$时,式(3)表示的计算完全隐藏访存延迟的约束条件是否满足仅取决于计算中的矩阵规模 K. 随着矩阵规模的增大,计算可以更好地隐藏存储访问的延迟,协处理器的计算效率也进一步提升. 表 2 中存储访问延迟占矩阵乘总执行时间的比例验证了上述分析,即较大的矩阵规模对应较低的存储访问延迟占比.

4.4 功能正确性验证与硬件实现开销的评估

本文基于 Verilog 硬件描述语言实现了双精度浮点矩阵乘协处理器. 其中,线性阵列计算结构中各计算单元采用双缓冲优化技术,PE 数设置为 1024、局部存储器的深度设置为 2048.

对于矩阵乘协处理器功能正确性的验证,本文以 Linpack 性能测试程序 HPL[25] 随机生成的双精度浮点矩阵数据作为测试数据集,通过调用 OpenBLAS[26-27] 线性代数计算函数库实现了双精度浮点矩阵乘运算核心. 本文将 OpenBLAS 的计算结果与矩阵乘协处理器的仿真计算结果进行了比对,从而验证了协处理器功能实现的正确性.

对于矩阵乘协处理器硬件开销的评估,本文基于 Synopsys Design Compiler 设计工具在 1.5 GHz、主流工艺下对协处理器的硬件设计进行了逻辑综合,综合过程中的参数配置如表 3 所示:

Table 3 Configurations for Synopsys Design Compiler

表 3 逻辑综合的参数设置

Parameters	Configuration
period	0.66
setup	0.083
hold	0.150
route_type	Mesh
max_layer	M4
clk_uncertainty	0.06
clk_transition	0.04

根据 Design Compiler 输出的综合结果,矩阵乘协处理器的硬件设计完全满足 1.5 GHz 频率下的时序要求. 其功耗为 38.57 W,总面积为 116.26 mm^2. 其中,组合逻辑的面积为 16.78 mm^2,非组合逻辑的面积为 99.48 mm^2.

本文将矩阵乘协处理器与采用同等工艺的 NVIDIA K40 GPGPU[19] 芯片进行了对比,比较的内容包括时钟频率、峰值计算性能,进行矩阵乘运算的实际性能和效率以及硬件开销等,结果如表 4 所示. 针对双精度浮点矩阵乘运算,本文协处理器的实际性能是 K40 GPGPU 芯片的 1.95 倍,而面积开销仅为后者的 21.05%(不考虑 GPGPU 芯片中 PCIe、图形显示和存储控制器等接口的硬件开销).

Table 4 Coprocessor versus K40 GPGPU on Performance and Hardware Overhead

表 4 协处理器与 K40 GPGPU 计算性能和硬件开销的对比

Design Parameters	K40	Coprocessor
Base Clock/MHz	745	1500
Boost Clock/MHz	875	
Peak Performance/GFLOPS	1680	3072
Matrix Multiplication Efficiency/%	93	99
Sustainable Performance	1562	3041
Area/mm^2	551	116

5 结束语

矩阵乘运算广泛应用于科学与工程领域. 因计算复杂度较高,且在 CPU,GPGPU,FPGA 等现有计算平台上的性能和效率受限,双精度浮点矩阵乘运算往往成为大规模数值计算应用的性能瓶颈. 近年来,面向应用的硬件定制成为高性能、高效计算结构

设计的重要趋势.本文将这一结构设计方法应用到矩阵乘的加速计算中,设计了双精度浮点矩阵乘定制协处理器和基于协处理器的加速计算系统.

本文双精度浮点矩阵乘协处理器的设计以线性阵列计算结构为基础,应用了双缓冲技术并对存储访问调度进行了优化.同时,本文构建了针对协处理器的性能模型,并基于性能模型深入分析了协处理器的各结构设计参数对其计算性能和效率的影响.此外,本文还验证了矩阵乘协处理器功能实现的正确性,并在主流工艺下评估了其硬件开销.本文双精度浮点矩阵乘协处理器的实际计算性能可达 3 TFLOPS,计算效率最高可达 99%,有效实现了大规模数值计算应用的加速.与同等工艺的 GPGPU 计算平台相比,本文设计的矩阵乘协处理器在计算性能和硬件开销方面均具有明显优势.

本文探索了硬件定制的结构设计在高性能计算中的应用.后续研究将基于 BLAS 库和 Linpack 性能测试程序评估本文双精度浮点矩阵乘协处理器的应用对大规模计算系统性能和效率的提升效果;同时分析协处理器实际性能随阵列计算结构中计算单元数增加的可扩展性,从而进一步优化协处理器的结构设计.另外,协处理器的应用适应性也是未来研究工作的重点.

参　考　文　献

[1] Dongarra J J, Duff I S, Sorensen D C, et al. Numerical Linear Algebra on High-Performance Computers [M]. Philadelphia, PA: SIAM, 1998

[2] Dongarra J J, Luszczek P, Petitet A. The LINPACK benchmark: Past, present and future [J]. Concurrency and Computation: Practice and Experience, 2003, 15(9): 803-820

[3] Wang Endong, Zhang Qing, Shen Bo, et al. High-Performance Computing on the Intel® Xeon Phi™ [M]. Berlin: Springer, 2014

[4] IBM Corporation. Engineering and scientific subroutine library (ESSL) and parallel ESSL [EB/OL]. [2017-08-04]. https://www-03.ibm.com/systems/power/software/essl/

[5] Barrachina S, Castillo M, Igual F, et al. Evaluation and tuning of the level 3 CUBLAS for graphics processors [C] //Proc of the 22nd IEEE Int Parallel & Distributed Processing Symp. Piscataway, NJ: IEEE, 2008: 3103-3111

[6] Heinecke A, Vaidyanathan K, Smelyanskiy M, et al. Design and implementation of the Linpack benchmark for single and multi-node systems based on Intel Xeon Phi coprocessor [C] //Proc of the 27th IEEE Int Parallel & Distributed Processing Symp. Piscataway, NJ: IEEE, 2013: 126-137

[7] Oak Ridge National Laboratory. IBM Power8 CPU overview [EB/OL]. [2017-07-27]. https://www.olcf.ornl.gov/wp-content/uploads/2017/01/Summit/Dev_IBM-Power8-CPUs_Walkup.pdf

[8] Wang Shen, Qi Fengbing, Gu Hongfeng, et al. Linpack parallel performance model and its prediction [J]. Computer Engineering, 2012, 38(16): 81-84 (in Chinese)
(王申,漆锋滨,谷洪峰,等. Linpack 并行性能模型及其预测[J]. 计算机工程, 2012, 38(16): 81-84)

[9] Nowatzki T, Gangadhan V, Sankaralingam K, et al. Pushing the limits of accelerator efficiency while retaining programmability [C] //Proc of the 22nd Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2016: 27-39

[10] Singapura S G, Panangadan A, Prasanna V K. Performance modeling of matrix multiplication on 3D memory integrated FPGA [C] //Proc of the 29th IEEE Int Parallel & Distributed Processing Symp Workshops. Piscataway, NJ: IEEE, 2015: 154-162

[11] Kumar V B Y, Joshi S, Patkar S B, et al. FPGA based high performance double-precision matrix multiplication [J]. International Journal of Parallel Programming, 2010, 38(3): 322-338

[12] Wu Guiming. Parallel algorithms and architectures for matrix computations on FPGA [D]. Changsha: National University of Defense and Technology, 2011 (in Chinese)
(邬贵明. FPGA 矩阵计算并行算法与结构[D]. 长沙:国防科学技术大学, 2011)

[13] Wu Guiming, Dou Yong, Wang Miao. High performance and memory efficient implementation of matrix multiplication on FPGAs [C] //Proc of the 9th IEEE Int Conf on Field Programmable Technology. Piscataway, NJ: IEEE, 2010: 134-137

[14] Zhou Leitao, Tao Yaodong, Liu Sheng, et al. Research on systolic multiplication and technology based on FPGA [J]. Journal of Computer Science and Engineering, 2015, 37(9): 1632-1636 (in Chinese)
(周磊涛,陶耀东,刘生,等. 基于 FPGA 的 Systolic 乘法技术研究[J]. 计算机工程与科学, 2015, 37(9): 1632-1636)

[15] Jovanović Ž, Milutinović V. FPGA accelerator for floating-point matrix multiplication [J]. IET Computers & Digital Techniques, 2012, 6(4): 249-256

[16] Lei Yuanwu, Chen Xiaowen, Peng Yuanxi. A high energy efficiency FFT accelerator on DSP chip [J]. Journal of Computer Research and Development, 2016, 53(7): 1438-1446 (in Chinese)
(雷元武,陈小文,彭元喜. DSP 芯片中的高能效 FFT 加速器[J]. 计算机研究与发展, 2016, 53(7): 1438-1446)

[17] Qian Lei, Zhao Jinming, Peng Dajia, et al. Energy-efficient fingerprint matching based on reconfigurable micro server [J]. Journal of Computer Research and Development, 2016, 53(7): 1425-1437 (in Chinese)

(钱磊, 赵锦明, 彭达佳, 等. 基于可重构微服务器的高能效指纹比对方法[J]. 计算机研究与发展, 2016, 53(7): 1425-1437)

[18] Jouppi N P, Young C, Patil N, et al. In-datacenter performance analysis of a tensor processing unit [C] //Proc of the 44th IEEE Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2017: 1-12

[19] NVIDIA Corporation. Inside volta: The world's most advanced data-center GPU [EB/OL]. [2017-06-17]. https://devblogs. nvidia. com/parallelforall/inside-volta/

[20] Sze V, Chen Y H, Suleiman, A, et al. Hardware for machine learning: Challenges and opportunities [C] //Proc of the 30th IEEE Custom Integrated Circuits Conf. Piscataway, NJ: IEEE, 2017: 299-306

[21] Gupta S, Agrawal A, Gopalakrishnan K, et al. Deep learning with limited numerical precision [C] //Proc of the 32nd Int Conf on Machine Learning. New York: ACM, 2015: 1737-1746

[22] JDJEC Organization. DDR3 SDRAM standard [EB/OL]. [2017-06-18]. https://www. jedec. org/standards-documents/docs/jesd-79-3d

[23] Lee Y, Waterman A, Avizienis R, et al. A 45nm 1.3GHz 16.7 double-precision GFLOPS/W RISC-V processor with vector accelerators [C] //Proc of the 40th IEEE European Solid-State Circuit Conf. Piscataway, NJ: IEEE, 2014: 199-202

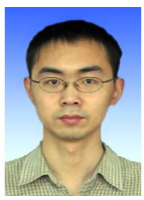
[24] Overton M L. Numerical Computing with IEEE Floating Point Arithmetic [M]. Philadelphia, PA: SIAM, 2001

[25] Barrett R F, Chan T, D'Azevedo E F, et al. Complex version of high performance computing LINPACK benchmark (HPL) [J]. Concurrency & Computation Practice & Experience, 2010, 22(5): 573-587

[26] Zhang Xianyi, Wang Qian, Zhang Yunquan. OpenBLAS: A high-performance BLAS library on Loongson 3A CPU [J]. Journal of Software, 2011, 22(2): 208-216 (in Chinese)
(张先轶, 王茜, 张云泉. OpenBLAS:龙芯 3A CPU 的高性能 BLAS 库[J]. 软件学报, 2011, 22(2): 208-216)

[27] GitHub. OpenBLAS: An optimized BLAS library [EB/OL]. [2017-08-04]. https://github. com/xianyi/OpenBLAS

Jia Xun, born in 1989. PhD candidate. Student member of CCF. His main research interests include high performance computing and processor micro-architecture.



Wu Guiming, born in 1981. PhD, engineer. His main research interests include high performance computing and processor micro-architecture. (wu. guiming@meac-skl. cn)

Xie Xianghui, born in 1958. PhD supervisor and professor. Senior member of CCF. His main research interests include computer architecture and parallel computing. (xie. xianghui@meac-skl. cn)

Wu Dong, born in 1972. PhD and senior engineer. His main research interests include high performance computer architecture and reconfigurable computing. (wu. dong@meac-skl. cn)