软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(3):1569−1584 [doi: 10.13328/j.cnki.jos.006811]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

国产 SW26010-Pro 处理器上 3 级 BLAS 函数众核并行优化*

胡 怡[1,2], 陈道琨[1,2], 杨 超[3], 马文静[1,2], 刘芳芳[1,2], 宋超博[4], 孙 强[4], 史俊达[4]

[1](中国科学院 软件研究所 并行软件与计算科学实验室, 北京 100190)

[2](中国科学院大学, 北京 100049)

[3](北京大学 数学科学学院, 北京 100871)

[4](国家并行计算机工程技术研究中心, 北京 100190)

通信作者: 杨超, E-mail: chao_yang@pku.edu.cn

摘 要: BLAS (basic linear algebra subprograms) 是最基本、最重要的底层数学库之一. 在一个标准的 BLAS 库中, BLAS 3 级函数涵盖的矩阵-矩阵运算尤为重要, 在许多大规模科学与工程计算应用中被广泛调用. 另外, BLAS 3 级属于计算密集型函数, 对充分发挥处理器的计算性能有至关重要的作用. 针对国产 SW26010-Pro 处理器研究 BLAS 3 级函数的众核并行优化技术. 具体而言, 根据 SW26010-Pro 的存储层次结构, 设计多级分块算法, 挖掘矩阵运算的并行性. 在此基础上, 基于远程内存访问 (remote memory access, RMA) 机制设计数据共享策略, 提高从核间的数据传输效率. 进一步地, 采用三缓冲、参数调优等方法对算法进行全面优化, 隐藏直接内存访问 (direct memory access, DMA) 访存开销和 RMA 通信开销. 此外, 利用 SW26010-Pro 的两条硬件流水线和若干向量化计算/访存指令, 还对 BLAS 3 级函数的矩阵-矩阵乘法、矩阵方程组求解、矩阵转置操作等若干运算进行手工汇编优化, 提高了函数的浮点计算效率. 实验结果显示, 所提出的并行优化技术在 SW26010-Pro 处理器上为 BLAS 3 级函数带来了明显的性能提升, 单核组 BLAS 3 级函数的浮点计算性能最高可达峰值性能的 92%, 多核组 BLAS 3 级函数的浮点计算性能最高可达峰值性能的 88%.

关键词: BLAS 3 级; SW26010-Pro 众核处理器; 直接内存访问; 远程内存访问; 浮点计算效率

中图法分类号: TP303

中文引用格式: 胡怡, 陈道琨, 杨超, 马文静, 刘芳芳, 宋超博, 孙强, 史俊达. 国产SW26010-Pro处理器上3级BLAS函数众核并行优化. 软件学报, 2024, 35(3): 1569–1584. http://www.jos.org.cn/1000-9825/6811.htm

英文引用格式: Hu Y, Chen DK, Yang C, Ma WJ, Liu FF, Song CB, Sun Q, Shi JD. Many-core Parallel Optimization of Level-3 BLAS Function on Domestic SW26010-Pro Processor. Ruan Jian Xue Bao/Journal of Software, 2024, 35(3): 1569–1584 (in Chinese). http://www.jos.org.cn/1000-9825/6811.htm

Many-core Parallel Optimization of Level-3 BLAS Function on Domestic SW26010-Pro Processor

HU Yi[1,2], CHEN Dao-Kun[1,2], YANG Chao[3], MA Wen-Jing[1,2], LIU Fang-Fang[1,2], SONG Chao-Bo[4], SUN Qiang[4], SHI Jun-Da[4]

[1](Laboratory of Parallel Software and Computational Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

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

[3](School of Mathematical Sciences, Peking University, Beijing 100871, China)

[4](National Research Center of Parallel Computer Engineering and Technology, Beijing 100190, China)

Abstract: Basic linear algebra subprogram (BLAS) is one of the most basic and important math libraries. The matrix-matrix operations covered in the level-3 BLAS functions are particularly significant for a standard BLAS library and are widely employed in many large-

* 基金项目: 国家重点研发计划 (2020YFB0204601)
收稿时间: 2021-11-22; 修改时间: 2022-02-23, 2022-07-26; 采用时间: 2022-09-29; jos 在线出版时间: 2023-05-10
CNKI 网络首发时间: 2023-05-11

scale scientific and engineering computing applications. Additionally, level-3 BLAS functions are computing intensive functions and play a vital role in fully exploiting the computing performance of processors. Multi-core parallel optimization technologies are studied for level-3 BLAS functions on SW26010-Pro, a domestic processor. According to the memory hierarchy of SW26010-Pro, this study designs a multi-level blocking algorithm to exploit the parallelism of matrix operations. Then, a data-sharing scheme based on remote memory access (RMA) mechanism is proposed to improve the data transmission efficiency among CPEs. Additionally, it employs triple buffering and parameter tuning to fully optimize the algorithm and hide the memory access costs of direct memory access (DMA) and the communication overhead of RMA. Besides, the study adopts two hardware pipelines and several vectorized arithmetic/memory access instructions of SW26010-Pro and improves the floating-point computing efficiency of level-3 BLAS functions by writing assembly code manually for matrix-matrix multiplication, matrix equation solving, and matrix transposition. The experimental results show that level-3 BLAS functions can significantly improve the performance on SW26010-Pro by leveraging the proposed parallel optimization. The floating-point computing efficiency of single-core level-3 BLAS is up to 92% of the peak performance, while that of multi-core level-3 BLAS is up to 88% of the peak performance.

Key words: level-3 BLAS; SW26010-Pro many-core processor; direct memory access (DMA); remote memory access (RMA); floating point computing efficiency

BLAS (basic linear algebra subprograms) 是高性能领域重要的底层数学库之一[1,2], 其中 3 级 BLAS 函数在气象预报[3]、神经网络[4]、量子模拟[5]和石油勘探[6]等领域应用广泛. 特别地, 用于世界 TOP500 超级计算机排名的基准测试程序 HPL (high performance linpack) 极大程度依赖 BLAS 3 级函数的性能[7]. 因此, 研究 BLAS 3 级函数的并行算法和性能优化技术具有重要的实用价值和研究意义. BLAS 3 级函数是典型的计算密集型任务, 为获得高浮点计算效率, 必须充分挖掘处理器底层的硬件特性, 从而释放处理器的计算性能.

硬件厂商针对各自处理器平台提供了高度优化的 BLAS 库, 如, Intel 公司的 MKL[8], AMD 公司的 AOCL[9], NVIDIA 公司的 cuBLAS[10]. 此外, 诸多研究机构针对不同处理器的体系结构特点实现的 GotoBLAS[11]、ATLAS[12]、OpenBLAS[13]等开源数学库也对 BLAS 3 级函数进行了深度优化. SW26010-Pro 是我国自主研发的申威异构众核处理器, 相比于其他处理器, 它的体系结构复杂, 存储层次多, 为商用处理器定制的 3 级 BLAS 库无法在 SW26010-Pro 上运行. 现有开源数学库在设计时没有考虑异构众核处理器的特点, 其优化方案也不能充分发挥 SW26010-Pro 的浮点计算峰值性能. 因此, 针对申威众核处理器特有的体系结构设计高性能 BLAS 3 级函数库十分必要. SW26010-Pro 与早期的 SW26010 在硬件设计上有很大不同, 具体而言, SW26010-Pro 将寄存器通信机制调整为 RMA 通信机制, 增大了向量化宽度和 LDM (局部存储器) 空间, 还支持 LDM 空间的可变分配, 可将其部分配置成硬件自动管理的数据 Cache (高速缓冲存储器). 此外, SW26010-Pro 还提供了若干向量化计算/访存指令和支持所有核组内存共享的大共享模式. SW26010 上已有的优化方案[14]不足以挖掘 SW26010-Pro 的硬件特性. 因此, 迫切需要针对 SW26010-Pro 众核处理器为 BLAS 3 级函数设计新的并行算法和优化技术, 以充分发挥 SW26010-Pro 的计算性能.

本文实现了一套面向 SW26010-Pro 众核处理器的 BLAS 3 级函数并行算法和性能优化方法. 该算法根据 SW26010-Pro 的存储层次结构对矩阵进行多级分块, 将矩阵元素合理地布局到层次化存储结构上, 挖掘矩阵运算的并行性; 并根据分块矩阵与从核的映射方式, 设计了基于远程内存访问 (remote memory access, RMA) 机制的数据共享策略, 提高了从核间的数据传输效率. 此外, 本文还充分结合 SW26010-Pro 的硬件特性, 对 BLAS 3 级函数的并行算法进行了深度优化. 主要包括, 使用三缓冲技术, 隐藏访存开销和通信开销; 通过参数调优为 BLAS 3 级各个函数选择了最优的分块参数, 进一步隐藏了访存开销; 充分利用 SW26010-Pro 的硬件流水线和若干向量化计算/访存指令, 对 BLAS 3 级函数的矩阵-矩阵乘法、矩阵方程组求解、矩阵转置操作等若干运算进行了手工汇编优化, 提高了函数的浮点计算效率. 本文优化的单核组 BLAS 3 级函数浮点计算性能最高可达峰值性能的 92%, 多核组 BLAS 3 级函数浮点计算性能最高可达峰值性能的 88%.

1 背景介绍

1.1 BLAS 3 级函数简介

BLAS 3 级函数实现矩阵-矩阵运算, 矩阵均按照列主元格式存储, 主要包括 6 个函数 (GEMM、SYMM、

SYRK、SYR2K、TRMM、TRSM), 支持实数单/双精度、复数单/双精度 4 种数据类型, 访存复杂度为 O(n^2), 计算复杂度为 O(n^3), 属于计算密集型函数. 除 TRSM 以外, 其他 BLAS 3 级函数实现矩阵-矩阵乘法操作. 本文以通用矩阵-矩阵乘法 (general matrix-matrix multiplication, GEMM) 和三角矩阵方程组求解 (solving a triangular matrix equation, TRSM) 为例进行说明. GEMM 的计算过程见公式 (1):

$$C \leftarrow \alpha \times op(A) \times op(B) + \beta \times C \tag{1}$$

其中, α, β 是标量参数. A, B, C 是大小分别为 $m \times k$, $k \times n$, $m \times n$ 的矩阵, $op(X)$ 可为 X、X^{T}、X^{H}, 其中, X^{T} 为 X 的转置, X^{H} 为 X 的共轭转置.

TRSM 根据矩阵 A 和 B, 计算满足等式:

$$AX = \alpha B \ (\text{左乘形式})$$

或等式:

$$XA = \alpha B \ (\text{右乘形式})$$

的矩阵, 其中 α 是标量, 矩阵 A 为上 (下) 三角矩阵.

1.2 SW26010-Pro 众核处理器

SW26010-Pro 是具有主从异构架构的国产众核处理器, 每个处理器节点由 6 个核组 (CG) 和系统接口组成, 每个核组主要包括 1 个主核 (MPE) 和 1 个从核 (CPE) 阵列. 从核阵列由 64 个从核构成, 以 8×8 阵列方式排布的从核阵列为基本单位进行管理, 4 个邻近从核共享一个从核簇管理部件 (SCM), 如图 1 所示. 主核可运行单个进程, 并且可发起 64 个从核线程, 每个线程只可运行在唯一的从核上. 为方便描述, 我们使用 CPE(I, J) 表示位于从核阵列第 I 行第 J 列的从核. 此外, SW26010-Pro 众核处理器还提供了大共享模式, 在此模式中, 所有核组可共享 96 GB 的内存, 每个核组可绑定一个主核线程.

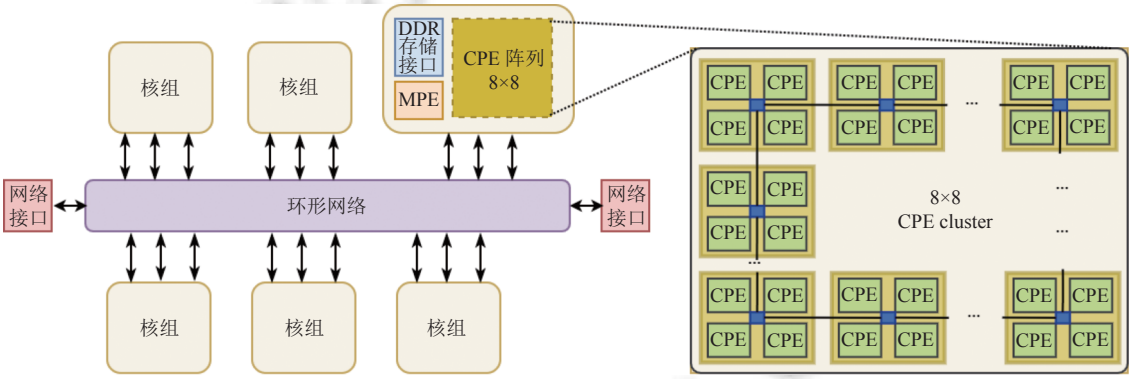


图 1 SW26010-Pro 架构以及从核阵列结构逻辑示意图

MPE 和 64 个 CPE 都采用我国自主研发的 SW64 指令集, 共享 16 GB 的主存, 每个 CPE 配置 256 KB 的局部存储器 (LDM), 该空间可配置成完全由软件管理, 也可将 32 KB 或 128 KB 配置成硬件自动管理的数据 Cache (高速缓冲存储器). 从核可通过发起异步 DMA (direct memory access, 直接内存访问) 的方式在主存和本地 LDM 之间传递数据, 数据的传输粒度以及是否连续会在很大程度上影响带宽. 64 个从核并发 DMA 操作可以提供 46 GB/s 的访存带宽, 不保证从核 DMA 调用间的数据一致性, 大共享模式中所有从核并发 DMA 操作可以提供 240 GB/s 的访存带宽.

从核可通过 RMA 机制实现本地 LDM 与其他从核 LDM 之间的数据传输, RMA 是核组内从核 LDM 之间进行的远程数据传输操作, 不同于 DMA, RMA 传输不支持跨步访问. RMA 为单边接口类型, 它由单个从核主动发起 RMA 数据传输, 其他相关从核通过判断回答字的值确定 RMA 操作是否完成. RMA 包含多种传输类型, 其中常用的是单从核模式和广播模式, 实现单个从核与单个/多个从核的数据交换, 支持阻塞和非阻塞方式. 所有 RMA 操

作均可异步进行, 操作的数据地址、数据长度必须 4 字节对齐.

SW26010-Pro 众核处理器的主核和从核均支持 SIMD (single instruction multiple data) 向量化指令, 包括向量化乘加指令等. 主核支持的 SIMD 处理长度为 256 位, 从核支持的 SIMD 处理长度为 512 位, 单/双精度的向量运算指令能同时处理 8 个元素. 此外, 从核支持 IEEE 754 标准的半精度、单精度和双精度浮点数据类型. 每个从核配置两条流水线, 其中一条用于浮点数运算, 另外一条用于访存操作. 从核阵列可集中 64 个从核的计算资源一起执行并行计算任务, 浮点计算峰值性能为 16 flop/cycle × 2.25 GHz × 64 = 2304 GFLOPs/s .

SW26010-Pro 众核处理器提供了若干向量化计算/访存指令. 在计算指令方面, vlenma 是浮点向量乘减指令, 它对任意 3 个向量操作数 A、B、C 执行如下运算:

$$C \leftarrow C - (A_i \cdot B),$$

其中, ·表示标量向量乘积符号, A_i 表示向量 A 的某个分量. vlenmas 是浮点向量乘减并存储指令, 它在 vlenma 指令的基础上还可将结果写回 LDM 中. 在访存指令方面, plut/pwrt (并行查/写表指令) 是向量化访存指令, 它可对 LDM 中若干不连续的地址进行读/写操作. plut/pwrt 的访存地址来自 LDM 的不同通道, 其中通道号为 LDM 地址的第 2–5 位.

1.3 相关工作

在高性能计算平台上有大量针对 BLAS 3 级函数的并行优化工作. 例如, Goto 等人提供了针对不同 CPU 的高性能 BLAS 3 级函数实现方案[15]. Wang 等人针对主流多核处理器研究了稠密线性代数子程序的优化技术, 能够直接为 BLAS 3 级函数生成高度优化的汇编代码[16]. Zhang 等人在龙芯 3A CPU 上优化了 BLAS 3 级函数[17]. CPU 上与 BLAS 3 级函数有关的优化工作采用了经典 3 层循环分块算法, 在算法实现时, 主要考虑了不同存储层次之间的数据移动开销是否可被浮点运算开销隐藏, 以及 TLB 不命中可能带来的性能损失, 还考虑了分块大小对 BLAS 3 级函数性能的影响. 对于配置多级缓存的 CPU 架构而言, 针对 BLAS 3 级函数的优化工作主要考虑了缓存的大小、TLB 的大小、缓存数据映射策略、计算和访存速率以及寄存器分块大小等因素.

随着 GPU 加速器的快速发展, BLAS 3 级函数在 GPU 上的优化工作引起了越来越多的关注. 例如, Nath 等人给出了在 Nvidia GPU 上实现高性能 BLAS 3 级函数的并行算法[18]. Wang 等人在多 GPU 平台利用逐块算法和 GPU 间的 P2P 通信, 实现了高度优化的 3 级 BLAS 库[19]. Igual 等人在 Nvidia GPU 上进一步优化了 cuBLAS 库中的 BLAS 3 级函数[20]. BLAS 3 级函数在 GPU 上的优化, 注重线程级并行性的挖掘以及 shared memory 的数据重用. 上述工作将矩阵 C 进行二维划分, 并将划分的子矩阵映射给对应的线程块进行更新. 线程块内的每个线程负责部分矩阵 A 和 B 元素的加载以及部分矩阵 C 元素的更新. 一个线程块内的所有线程可共享 shared memory 的矩阵元素. 上述工作对分块大小、线程块的数目以及寄存器分块大小等参数进行了细致调优, 还使用了双缓冲技术, 隐藏全局访存开销.

随着国产处理器的崛起, 设计并实现高性能 BLAS 3 级函数引起了学术界的广泛关注. 例如, 刘昊等人基于国产申威 1600 平台, 采用高效的分块算法和向量化指令、循环展开与软件流水、数据预取与数据重排等优化技术, 实现了 BLAS 3 级的 GEMM 函数[21]. Jiang 等人基于国产 SW26010 众核处理器的单核组, 优化了 BLAS 3 级的 GEMM 函数[14], 针对申威众核处理器存储结构的特点, 设计了 3 级分块算法, 基于 SW26010 的寄存器通信机制解决了从核间的数据交换问题, 采用主从核间的异步 DMA 机制设计了双缓冲策略, 进一步隐藏了 DMA 访存开销, 利用指令重排技术隐藏了片上局部访存和通信开销.

纵观高性能计算平台上的工作可以发现, 无论是 CPU 等同构计算平台, 还是 GPU、申威等异构计算平台, BLAS 3 级函数的高性能实现都包括分块算法设计和核心函数优化两部分. 我们秉承了这样的设计方法, 将 BLAS 3 级函数的高性能实现分为并行算法和优化方法两部分, 并行算法中的多级分块算法借鉴了 Jiang 等人[14]的分块思路. 在此基础上, 我们基于 SW26010-Pro 特有的 RMA 机制设计了数据共享策略, 提高了从核间的数据传输效率. 此外, 我们还采用三缓冲、参数调优等方法对算法进行了全面优化, 进一步隐藏了 DMA 访存开销和 RMA 通信开销. 特别地, 我们利用 SW26010-Pro 的两条硬件流水线和若干向量化计算/访存指令, 对 BLAS 3 级函数的矩

阵-矩阵乘法、矩阵方程组求解、矩阵转置操作等若干运算进行了手工汇编优化, 提高了函数的浮点计算效率. 子矩阵 $\delta C_{i,j}$ 的子块 ϵC 的计算过程如算法 1 所示.

算法 1. 子块 ϵC 的计算过程.

1. **for** r = 0 to 7 **do**
2. 列号为 r 的线程读取 $\epsilon A_{u,r}$, 行号为 r 的线程读取 $\epsilon B_{r,v}$
3. DMA wait
4. 列号为 r 的线程行广播 $\epsilon A_{u,r}$, 行号为 r 的线程列广播 $\epsilon B_{r,v}$
5. RMA wait
6. $\epsilon C_{u,v}$ += $\epsilon A_{u,r} \times \epsilon B_{r,v}$
7. **endfor**

2 并行算法

本文针对 BLAS 3 级函数, 提出了一套适用于 SW26010-Pro 众核处理器的并行算法. 该并行算法采用分块策略, 以充分利用 SW26010-Pro 各存储层次的硬件资源. 在后文中, 我们首先围绕 SW26010-Pro 众核处理器的单核组详细介绍了矩阵-矩阵乘法操作的并行算法. TRSM 在矩阵-矩阵乘法操作的基础上还包含回代求解操作, 该操作的算法我们在第 2.2 节介绍. 在第 2.3 节我们将介绍将 BLAS 3 级函数拓展到多核组的并行算法.

2.1 矩阵-矩阵乘法的并行算法

我们针对矩阵-矩阵乘法操作, 设计了多级分块算法, 将矩阵元素合理地布局到 SW26010-Pro 的不同存储层次上 (包括主内存、LDM 和寄存器). 在此基础上, 我们设计了数据共享策略, 以满足从核的计算需求, 提高从核间的数据传输效率. 在后文中, 我们以矩阵 A、B 均为非转置的 GEMM 为例详细介绍了矩阵-矩阵乘法操作的并行算法.

图 2 描绘了 GEMM 的分块算法. 如图 2 的第 1 行所示, 矩阵 C、矩阵 A 和矩阵 B 被划分成大小分别为 $b_m \times b_n$, $b_m \times b_k$ 和 $b_k \times b_n$ 的子矩阵, 划分后原矩阵变成规模为 $M \times N$, $M \times K$ 和 $K \times N$ 的分块矩阵, 其中, $M = m/b_m$, $N = n/b_n$, $K = k/b_k$. 矩阵 C 的子矩阵按照 *N*-*M*-*K* 的循环次序进行更新, 如公式 (2) 所示:

$$\delta C_{i,j} \leftarrow \delta C_{i,j} + \alpha \times \sum_{l=0}^{K-1} (\delta A_{i,l} \times \delta B_{l,j}),\ 0 \leqslant i \leqslant M-1, 0 \leqslant j \leqslant N-1 \tag{2}$$

其中, $\delta X_{i,j}$ 表示分块矩阵 X 的第 i 行第 j 列的子矩阵, 算法复用子矩阵 $\delta C_{i,j}$ 的数据. 若子矩阵的规模小于分块规模, 本文以填 0 的形式对其进行扩充. 从核阵列完成公式 (2) 的矩阵乘法运算 $(\delta A_{i,l} \times \delta B_{l,j})$, 并将运算结果累加至 $\delta C_{i,j}$. 由于从核的 LDM 空间有限, 我们将 $\delta C_{i,j}$ 、 $\delta A_{i,l}$ 和 $\delta B_{l,j}$ 进一步划分成大小分别为 $p_m \times p_n$, $p_m \times p_k$ 和 $p_k \times p_n$ 的子块, 其中, $p_m = b_m/8$, $p_k = b_k/8$, $p_n = b_n/8$. 本文用符号 $\epsilon X_{u,v}$ 表示子矩阵 $\delta X_{i,j}$ 的第 u 行第 v 列的子块, 并将 $\epsilon X_{u,v}$ 映射给从核 CPE(u, v).

如图 2 的第 2 行所示, 公式 (2) 的矩阵乘法运算 $(\delta A_{i,l} \times \delta B_{l,j})$ 可以表示为 $\delta A_{i,l}$ 的列块与 $\delta B_{l,j}$ 的行块的外积形式. 从核阵列在更新 $\delta C_{i,j}$ 的过程中, 共包含 8 次外积操作, 每次外积操作, 每个从核计算的是 ϵC 的一个部分和. 每个列块或行块包含 8 个子块, 因此, 外积操作包含 64 个矩阵-矩阵乘运算, 由 64 个从核并行计算完成. 为满足从核外积运算的数据需求, 提高从核间的数据传输效率, 我们设计了基于 RMA 操作的数据共享策略. 当执行第 r 次外积操作时 ($0 \leqslant r \leqslant 7$), 从核 CPE(u, v) 需要计算 $\epsilon A_{u,r}$ 与 $\epsilon B_{r,v}$ 的乘积, 并将结果累加至 $\epsilon C_{u,v}$. 为了满足计算需求, 从核通过行/列广播的 RMA 通信机制获取其他从核上的数据, 共包含 8 次 RMA 广播操作, 由第 0–7 行/列的从核依次进行广播, 如算法 1 所示. 位于第 r 列 (行) 的从核通过 DMA 读取 $\epsilon A_{u,r}$ $(\epsilon B_{r,v})$, 并将其行 (列) 广播到同行 (列) 的其他从核. 图 3 和图 4 描绘了算法 1 的某一次迭代中各个从核的数据收发状态.

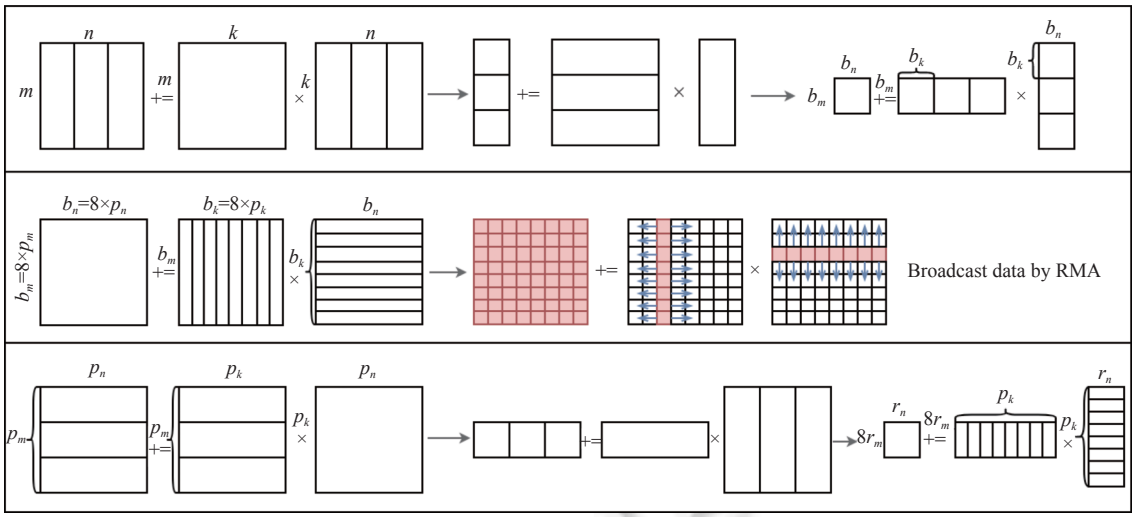


图 2 多级分块算法

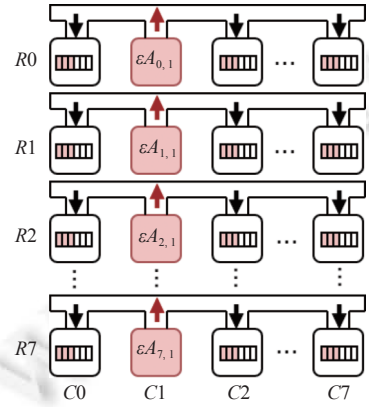


图 3 CPE(*, C1) 行广播矩阵 A 元素

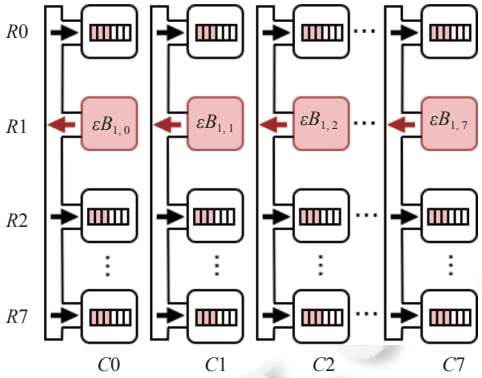


图 4 CPE($R1$, *) 列广播矩阵 B 元素

为提高从核的浮点计算效率, 我们采用汇编实现外积操作中的矩阵-矩阵乘运算 (见第 3.3.1 节). 在实现中, 我们将、和继续划分成大小分别为、和的计算子块, 如图 2 的第 3 行所示. 其中, r_m 和 r_n 分别表示每个从核存储矩阵 A 和矩阵 B 元素所用的寄存器数目. SYMM、TRMM、SYRK、SYR2K 的计算流程与 GEMM 相似, 采用的并行算法与其一致. 但它们的矩阵存储方式与 GEMM 有所不同, 矩阵分块后产生的某些以及为三角矩阵或对称矩阵, 主存中的数据仅包含该矩阵的上/下三角部分, 因此, 在计算前我们需要将其补全.

2.2 TRSM 的并行算法

TRSM 采用的并行算法与 GEMM 大致相同. 若矩阵 A 的对角元素不为 0, 在图 2 第 1 行的分块方式下, 下三角左乘形式的 TRSM 可用公式 (3) 表示. 我们将对角线部分的 TRSM 进一步划分, 子块的计算模式与公式 (3) 类似, 其中非对角线部分仍然采用 GEMM 的计算方式, 对角线部分 TRSM 的计算方式需要单独实现 (见第 3.3.2 节), 如图 5 所示. 若对角线部分的子块 $\epsilon A_{u,u}$ 的规模小于分块规模, 本文以填 0 的形式对其进行扩充, 并将扩充部分的对角元素置为 1. 根据 TRSM 问题的形式 (左乘或右乘), 在计算前, 我们将 $\epsilon A_{u,u}$ 的对角元素替换为它的倒数或负倒数.

$$\delta X_{i,j} \leftarrow \delta A_{i,i}^{-1} \times \left(\delta B_{i,j} - \sum_{0 \leqslant l < j} \delta A_{i,l} \times \delta X_{l,j} \right) \tag{3}$$

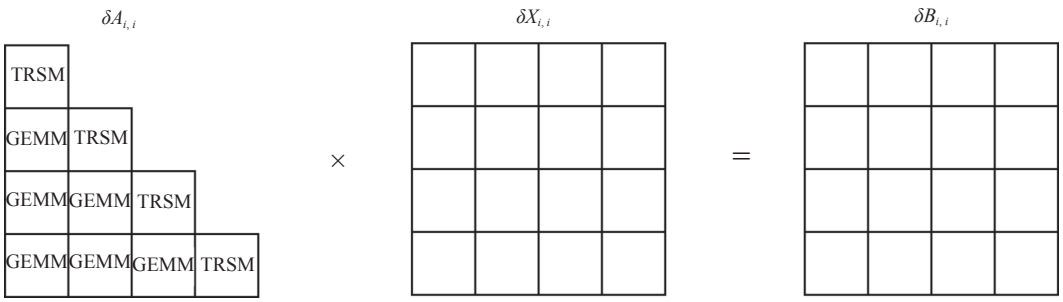


图 5 TRSM 算法

2.3 BLAS 3 级函数的多核组并行

针对多核组的情况, 我们采用 SW26010-Pro 处理器提供的大共享模式 (见第 1.2 节) 实现 BLAS 3 级函数的多核组并行. 在后文中, 我们首先介绍矩阵-矩阵乘法操作的多核组并行算法, TRSM 在矩阵-矩阵乘法操作的基础上还包含回代求解操作, 该操作的多核组并行算法我们单独介绍.

我们以矩阵 A、B 均为非转置形式的 GEMM 为例详细介绍矩阵-矩阵乘法操作的多核组并行算法. 在多核组层面, 我们将矩阵 C 进行二维划分, 划分成若干子矩阵, 在这种分块方式下, 子矩阵 $C_{i,j}$ 可按照如下公式更新:

$$C_{i,j} \leftarrow C_{i,j} + \alpha \times \sum_{l=0}^{k} (A_{i,l} \times B_{l,j}).$$

由上式可知, $C_{i,j}$ 的运算仍然是一个 GEMM 问题, 因此, 我们将 $C_{i,j}$ 的运算分配给每个核组, 由每个核组直接调用单核组的 GEMM 完成计算. SYMM、TRMM、SYRK、SYR2K 的计算流程与 GEMM 相似, 采用的多核组并行算法与其一致.

在多核组层面, 我们将左乘形式的 TRSM 的右端项矩阵 B 进行一维划分, 划分成若干列块, 将右乘形式的 TRSM 的右端项矩阵 B 划分成若干行块, 在这种分块方式下, 未知量矩阵 X 的列块或行块 X_i 可按照如下公式更新:

$$\begin{cases} X_i \leftarrow A^{-1} \times B_i (\text{左乘形式}) \\ X_i \leftarrow B_i \times A^{-1} (\text{右乘形式}) \end{cases}.$$

由上式可知, X_i 的运算仍然是一个 TRSM 问题, 因此, 我们将 X_i 的运算分配给每个核组, 由每个核组直接调用单核组的 TRSM 完成计算.

3 优化方法

为了进一步提升 BLAS 3 级函数的性能, 我们使用了 3 个重要的优化技术, 主要包括三缓冲技术、参数调优、以及汇编实现.

3.1 三缓冲技术

为了进一步隐藏算法 1 的 DMA 访存开销和 RMA 通信开销, 我们使用了三缓冲技术, 来并行发起 DMA、RMA 和计算操作, 使访存资源、通信资源和计算资源同时被调动, 达到掩盖 DMA 访存开销和 RMA 通信开销的目的. 使用三缓冲技术后的 ϵC 的计算步骤如算法 2 所示, 其中, 第 7 行的 DMA 操作、第 8 行的 RMA 操作以及第 9 行的计算可以异步进行, 掩盖了访存和通信开销.

算法 2. 使用三缓冲技术后 ϵC 的计算步骤.

1. 列号为 0 的线程读取 $\epsilon A_{u,0}$, 行号为 0 的线程读取 $\epsilon B_{0,v}$
2. DMA wait

3. 列号为 1 的线程读取 $\epsilon A_{u,1}$, 行号为 1 的线程读取 $\epsilon B_{1,v}$
4. 列号为 0 的线程行广播 $\epsilon A_{u,0}$, 行号为 0 的线程列广播 $\epsilon B_{0,v}$
5. DMA wait, RMA wait
6. **for** $r=2$ to 7 **do**
7. 列号为 r 的线程读取 $\epsilon A_{u,r}$, 行号为 r 的线程读取 $\epsilon B_{r,v}$
8. 列号为 $(r-1)$ 的线程行广播 $\epsilon A_{u,(r-1)}$, 行号为 $(r-1)$ 的线程列广播 $\epsilon B_{(r-1),v}$
9. $\epsilon C_{u,v}$ += $\epsilon A_{u,(r-2)} \times \epsilon B_{(r-2),v}$
10. DMA wait,RMA wait
11. **endfor**
12. 列号为 $(r-1)$ 的线程行广播 $\epsilon A_{u,(r-1)}$, 行号为 $(r-1)$ 的线程列广播 $\epsilon B_{(r-1),v}$
13. $\epsilon C_{u,v}$ += $\epsilon A_{u,(r-2)} \times \epsilon B_{(r-2),v}$
14. RMA wait

3.2 参数调优

BLAS 3 级函数的分块算法涉及寄存器级别、LDM 级别、主内存级别的分块, 分别对应分块参数 (r_m、r_n)、(p_m、p_n、p_k)、(b_m、b_n、b_k). 其中, (r_m、r_n) 受寄存器数量的限制, 决定汇编实现中计算子块的大小 (见前文图 2 的第 3 行), 影响计算指令和访存指令的比例, 进而决定手工汇编优化中访存开销能否被计算时间隐藏, 从而影响单从核的浮点计算效率. (p_m、p_n、p_k) 受 LDM 空间的限制, 决定 LDM 中子块的大小 (见前文图 2 的第 2 行), 进而决定三缓冲流水中 DMA 访存开销和 RMA 通信开销能否被计算时间隐藏, 从而影响函数整体的性能. 因此, 本文通过分析和实验, 对寄存器级别、LDM 级别的分块参数 (r_m、r_n)、(p_m、p_n、p_k) 进行了调优, 保证访存开销在层次化的存储结构中被计算隐藏. 此外, 由前文图 2 的第 2 行可知, 主内存级别的分块参数 $b_m=8\times p_m$、$b_n=8\times p_n$、$b_k=8\times p_k$.

3.2.1 寄存器级别的参数调优

在计算过程中, 我们使用 r_m 和 r_n 个寄存器分别存储矩阵 A 和矩阵 B 元素, $r_m\times r_n$ 个寄存器存储累加的矩阵 C 元素. 本文使用 vlenma 指令对矩阵-矩阵乘运算进行向量化加速. 由于 vlenma 指令执行一次需要 8 拍, 为了最大化浮点运算流水线的效率, 本文需要至少 8 条不相关的 vlenma 指令掩盖该指令的执行延迟, 所以:

$$r_m\times r_n \geqslant 8.$$

本文在考虑向量寄存器资源以及计算访存重叠等因素后, 满足条件的 r_m 和 r_n 组合有: (2, 4), (4, 2), (2, 5), (5, 2). 通过测试以上的 r_m 和 r_n 组合, 我们发现当 $r_m=2, r_n=4$ 时, 计算函数的效率最高, 我们选择该组合作为寄存器级别的分块参数.

3.2.2 LDM 级别和主内存级别的参数调优

为了隐藏 DMA 访存开销和 RMA 通信开销, 从核的计算时间需要大于其访存时间, 所以如下不等式成立:

$$\frac{2\times p_m\times p_n\times p_k}{2304\ \text{GFLOPs/s}} \geqslant \frac{(p_m\times p_k+p_k\times p_n+2\times p_m\times p_n)\times sizeof(double)}{46\ \text{GB/s}}.$$

上式表明, 当 $p_m\times p_n\times p_k$ 足够大时, 计算能够掩盖访存开销. 因为寄存器级别的分块大小 $r_m=2, r_n=4$, 所以, p_m 应该是 2×8 的倍数, p_n 应该是 4 的倍数. 又由于一次 DMA 事务最小数据单元是 256 B, 为了保证 DMA 操作的效率, p_k 应该被设置成 32 的倍数 (以双精度为例). 通过测试满足以上约束条件, 且在 LDM 空间限制内的不同参数组合, 我们发现当 $p_m=p_n=p_k=64$ 时, BLAS 3 级函数性能最优. 因此, 本文选择该组合作为 LDM 级别的分块参数, 相应地, 主内存级别的分块参数, $b_m=b_n=b_k=512$.

3.3 汇编实现

BLAS 3 级函数是计算密集型问题, 高性能 BLAS 3 级函数的浮点计算性能应接近机器的峰值性能, 手工汇编

优化是 BLAS 3 级函数在该平台上获取高性能的关键. 为此, 我们有必要充分利用 SW26010-Pro 众核处理器底层的两条硬件流水线和若干向量化计算/访存指令, 挖掘指令级并行, 提高 BLAS 3 级函数的浮点计算效率. 我们详细介绍了矩阵-矩阵乘运算、矩阵方程组求解、矩阵转置操作的汇编实现.

3.3.1 矩阵-矩阵乘运算的汇编实现

矩阵-矩阵乘法的汇编完成如下运算:

$$\epsilon C \leftarrow \epsilon C + \alpha(\epsilon A \times \epsilon B),$$

其中, 子块 ϵA , ϵB , ϵC 是规模均为 64×64 的矩阵, 该运算采用 p_m-p_n-p_k 的循环次序, 对矩阵 ϵC 的每个计算子块 $\sigma C_{i,j}$ 进行如下操作:

$$\sigma C_{i,j} \leftarrow \sigma C_{i,j} + \alpha \sum_{l=0}^{7}(\sigma A_{i,l} \times \sigma B_{l,j}),\ 0 \leqslant i \leqslant 3, 0 \leqslant j \leqslant 15 \tag{4}$$

其中, $\sigma C_{i,j}$ 是子块 ϵC 的第 i 行第 j 列的计算子块, 规模为 16×4 , $\sigma A_{i,l}$ 、 $\sigma B_{l,j}$ 是计算子块 σA 、 σB 的子矩阵, 规模分别为 16×8 、 8×4 . p_k 层循环完成公式 (4) 中求和部分的运算 $\sum_{l=0}^{7}(\sigma A_{i,l} \times \sigma B_{l,j})$, 本文采用 vlenma 指令实现公式 (4) 中的矩阵乘法运算 $(\sigma A_{i,l} \times \sigma B_{l,j})$. 如图 6 所示, 如果用 T 表示该矩阵乘法运算的结果, 我们将矩阵 $\sigma A_{i,l}$ 划分成向量 $A_{u0} - A_{u7}$ ($0 \leqslant u \leqslant 1$), 将矩阵 $\sigma B_{l,j}$ 划分成向量 B_v ($0 \leqslant v \leqslant 3$), 将矩阵 T 划分成向量 $T_{u0} \sim T_{u3}$ ($0 \leqslant u \leqslant 1$), 在 vlenma 指令的计算模式下, 向量 $T_{u0} - T_{u3}$ 可表示为如下形式:

$$-T_{uv} \leftarrow ((((0 - B_{v,0} \cdot A_{u0}) - B_{v,1} \cdot A_{u1}) - B_{v,2} \cdot A_{u2}) - \ldots - B_{v,7} \cdot A_{u7}) \tag{5}$$

公式 (5) 的 $B_{v,0} - B_{v,7}$ 表示向量 B_v 的第 $0-7$ 个分量. 在 p_k 层循环最后一次迭代时, 本文使用 vlenmas 指令, 将 p_k 层循环的计算结果与标量 α 相乘, 并累加至矩阵 $\sigma C_{i,j}$. 该指令计算完成后, 会自动将矩阵 $\sigma C_{i,j}$ 的结果写回 LDM.

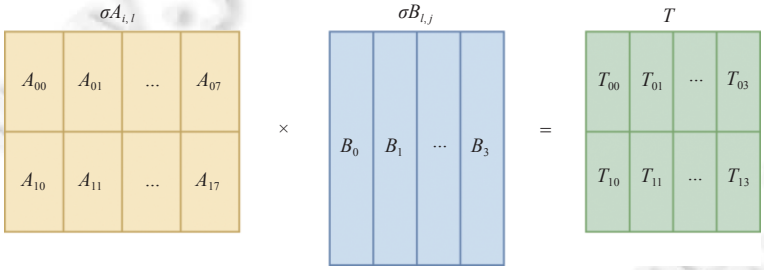


图 6 计算子块的乘法运算

公式 (5) 是对 8 个向量 (T_{uv}) 的操作, 每个向量的标量-向量乘操作是不相关的, 它们按 B_v 的分量可以分成 8 组, 每组包含 8 个标量-向量乘操作. 为了隐藏访存延迟, 我们将浮点向量乘减指令 (vlenma) 和取数指令 (vldd) 成对排列, 考虑到 SW26010-Pro 众核处理器双发射的特点, 在实现中, 我们还插入了若干空指令 nop, 避免乱序发射对流水线的干扰. 指令重排后的部分汇编指令如算法 3 所示. 其中, 每个向量各占用一个向量寄存器. 通过测试, 采用汇编实现的矩阵-矩阵乘运算的浮点计算效率可达机器单从核峰值性能的 97%, 相比未进行向量化的矩阵-矩阵乘运算加速了约 157 倍.

算法 3. p_k 层循环部分汇编指令.

1. vlenma $B_{0,0}$, A_{00}, T_{00}, T_{00}; vldd A_{01};
2. vlenma $B_{0,0}$, A_{10}, T_{10}, T_{10}; vldd A_{11};
3. vlenma $B_{1,0}$, A_{00}, T_{01}, T_{01}; nop;
4. vlenma $B_{1,0}$, A_{10}, T_{11}, T_{11}; nop;
5. vlenma $B_{2,0}$, A_{00}, T_{02}, T_{02}; nop;

6. vlenma $B_{2,0}$, A_{10}, T_{12}, T_{12}; nop;
7. vlenma $B_{3,0}$, A_{00}, T_{03}, T_{03}; nop;
8. vlenma $B_{3,0}$, A_{10}, T_{13}, T_{13}; nop;

3.3.2 矩阵方程组求解的汇编实现

TRSM 对角线部分求解的是一个规模为 64×64 的稠密三角矩阵方程组问题, 该问题可进一步划分为规模为 8×8 的矩阵-矩阵乘运算 (非对角线部分) 以及三角矩阵方程组子问题 (对角线部分). 非对角线部分本文采用第 3.3.1 节描述的方式进行向量化, 本节主要描述如何利用向量化指令, 加速对角线部分三角矩阵方程组问题的求解操作.

左乘形式的 TRSM 子问题由 8 个独立的三角线性方程组问题:

$$AX_i = B_i,\ i = 0,\ldots,7$$

构成, 其中, X_i (B_i) 为未知量矩阵 (右端项矩阵) 的第 i 个列向量. 不失一般性, 本文假设在左乘 TRSM 问题中, 矩阵 A 为下三角矩阵. 矩阵 A 对角线部分存储的是对角元素的倒数, 于 TRSM 的预处理步骤中提前计算 (见第 2.2 节). 如图 7 所示, 在上述三角线性方程组的回代求解过程 (back substitution) 中, 每个分量 $X_{i,l}$ ($l = 0,\ldots,7$) 的计算过程分为 3 个步骤.

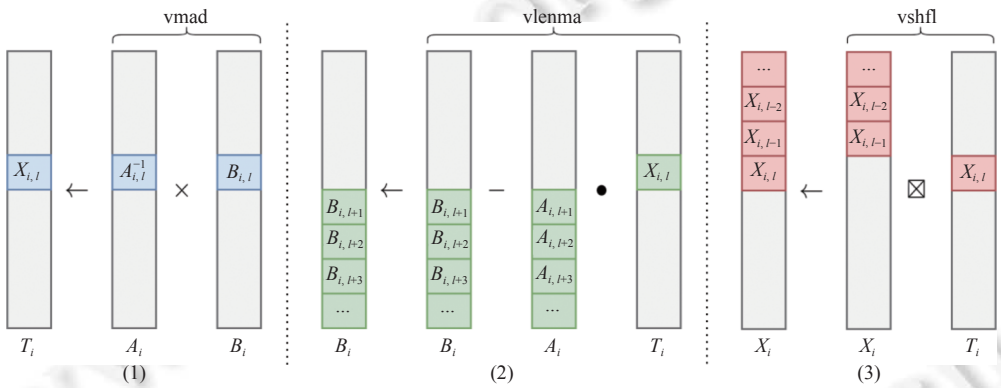


图 7 左乘形式的 TRSM 问题, 对角线部分的向量化回代求解计算步骤

(1) 通过浮点向量乘加指令 (vmad), 根据对角元素 $A_{i,l}$ 以及右端项元素 $B_{i,l}$, 计算得到分量 $X_{i,l}$, 并将结果存入临时浮点向量 T_i.

(2) 通过浮点向量乘减指令 (vlenma), 根据分量 $X_{i,l}$ 以及三角矩阵列向量 A_i, 更新右端项向量 B_i.

(3) 利用向量混洗指令 (vshfl), 从临时向量 T_i 中提取分量 $X_{i,l}$, 并存入解向量 X_i 的对应位置.

图 7 中, $A_{i,l}$ 表示矩阵 A 第 i 个列向量的第 l 个分量. 由于左乘 TRSM 问题需要同时回代求解 8 个线性方程组, 其计算步骤可产生足够多的向量化计算/混洗操作, 用于隐藏 vmad/vlenma/vshfl 指令的执行延迟. 若矩阵 A 为上三角矩阵, 可利用 vshfl 指令, 将向量 A_i / B_i / X_i 各分量的次序翻转后, 采用与上述步骤类似的方式实现向量化加速.

对于右乘形式的 TRSM 子问题:

$$XA = B,$$

若矩阵 A 为上三角矩阵, 可利用 vlenma 指令的计算模式, 将其解向量 X_i ($i = 0,\ldots,7$) 表示如下:

$$\begin{cases} X_0 \leftarrow 0-(-A_{0,0})^{-1}\cdot B_0 \\ X_1 \leftarrow 0-(-A_{1,1})^{-1}\cdot (B_1 - A_{1,0}\cdot \mathrm{X}_0) \\ X_2 \leftarrow 0-(-A_{2,2})^{-1}\cdot ((B_1 - A_{2,0}\cdot X_0) - A_{2,1}\cdot X_1) \\ \vdots \\ X_7 \leftarrow 0-(-A_{7,7})^{-1}\cdot (((B_7 - A_{7,0}\cdot X_0) - A_{7,1}\cdot X_1) - \ldots - A_{7,6}\cdot X_6) \end{cases}.$$

在右乘 TRSM 问题中, 矩阵 A 对角线部分存储的是对角元素的负倒数, 于 TRSM 的预处理步骤中提前计算(见第 2.2 节). 对右乘 TRSM 子问题的矩阵 A 为下三角矩阵的情形, 本文同样采用翻转各分量次序的方法.

3.3.3 矩阵转置操作的汇编实现

在公式 (1) 中, 若 $op(A)$ 为 A^{T} / A^{H}, 从核进行计算前, 需要对子块 ϵA 进行转置操作. 而矩阵转置操作耗时较多, 我们充分利用 SW26010-Pro 提供的并行查写表指令 (plut/pwrt), 实现向量化处理对其进行加速. 利用 plut 和 pwrt 指令, 我们设计了对规模为 64×64 的矩阵进行 out-of-place 转置操作的算法. 我们将原矩阵划分成多个 8×8的子矩阵, 将他们依次从原矩阵取出, 进行转置操作后, 再将他们写回目的矩阵. 子矩阵的转置过程如图 8 所示, 我们首先利用并行查表指令 (plut) 将矩阵元素加载到向量寄存器中, 读取的矩阵元素来自矩阵的不同行不同列, 避免通道冲突. 然后利用向量混洗指令 (vshfl) 将矩阵元素进行排列, 最后利用并行写表指令 (pwrt) 将矩阵元素写回 LDM 中. 相比于未进行向量化的矩阵转置操作, 矩阵转置操作的时间从约 597 拍降为 139 拍.

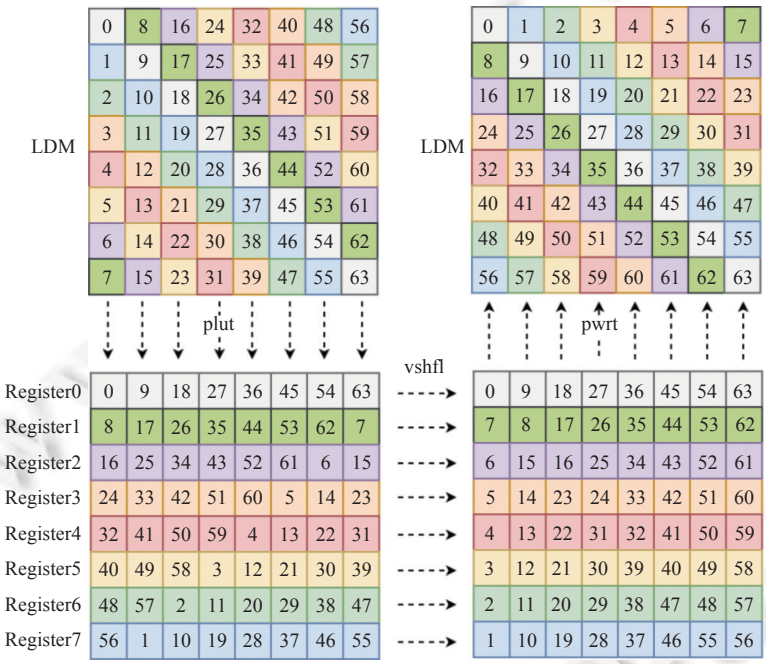


图 8 采用 plut 和 pwrt 指令实现的矩阵转置示意图

4 实验结果及分析

我们采用 SW26010-Pro 众核处理器作为测试平台, 测试本文优化的 BLAS 3 级函数在该平台上的性能. 我们首先测试了本文的几种优化方法分别带来的性能效果. 然后, 我们展示了每个 3 级函数的性能, 验证了本文所提出的并行优化算法有效性. 实验中的数据类型为双精度.

4.1 对比实验

我们以矩阵为非转置形式的 GEMM 和上三角右乘形式的 TRSM 作为实验对象, 采用 SW26010-Pro 众核处理器的一个核组作为测试平台, 集中测试了方阵乘法的优化算法性能, 并测试了几种优化方法分别带来的性能提升效果. 为了进一步分析前文中的各种优化手段对 GEMM 性能的影响, 我们实现了以下几种不同版本的 GEMM. 如下所述.

◆ BASELINE: 基于分块算法的简单实现, 未采用其他优化手段, 矩阵元素均通过 DMA 机制获取.

◆ GEMM KER: 在 BASELINE 的基础上, 加入矩阵-矩阵乘运算的汇编实现.

◆ RMA KER: 在 GEMM KER 的基础上, 加入基于 RMA 机制的数据共享策略.

◆ TB: 在 RMA KER 的基础上, 使用三缓冲技术进行优化.

图 9 对比了 4 种 GEMM 版本的性能. 从图中可看出, GEMM KER 版本相较于 BASELINE 版本性能有了显著提升. 主要是因为在 GEMM KER 版本中, 我们利用 SW26010-Pro 的向量指令, 通过排布汇编语句, 充分利用向量部件以及硬件流水线, 使采用汇编实现的矩阵-矩阵乘运算的浮点计算效率达到了单从核峰值性能的 97%, 较非向量化版本提升了 157 倍. 而在 GEMM KER 版本中, DMA 传输开销仍占有很大比例, 因此, 此时 GEMM 函数的整体性能提升了 20 倍, 为峰值性能的 13%. 在 RMA KER 版本中, 我们使用数据共享策略, 使从核可以通过 RMA 机制共享数据, 大大减少了从核的访存量, 降低了 DMA 传输开销, 使 GEMM 整体性能提升了 3.7 倍, 为峰值性能的 48%. 在 TB 版本中, 我们充分利用 DMA、RMA 和计算互不依赖的特性, 采用了三缓冲技术, 使 DMA 访存开销和 RMA 通信开销隐藏在计算时间内, GEMM 的整体性能达到峰值性能的 92%, 接近矩阵-矩阵乘运算的浮点计算效率 (97%).

为了进一步分析上文中的各种优化手段对 TRSM 性能的影响, 我们实现了以下几种不同版本的 TRSM.

◆ BASELINE: 基于分块算法的简单实现, 未采用其他优化手段, 矩阵元素均通过 DMA 机制获取.

◆ GEMM KER: 在 BASELINE 的基础上, 加入矩阵-矩阵乘运算的汇编实现.

◆ RMA KER: 在 GEMM KER 的基础上, 加入基于 RMA 机制的数据共享策略.

◆ TRSM KER: 在 RMA KER 的基础上, 加入矩阵方程组求解的汇编实现.

◆ TB: 在 TRSM KER 的基础上, 使用三缓冲技术进行优化.

图 10 对比了 5 种 TRSM 版本的性能, 从图中可看出, GEMM KER 版本相较于 BASELINE 版本性能有了显著提升. 在 GEMM KER 版本中, 我们利用向量化机制为 TRSM 非对角线部分的 GEMM 操作加入了矩阵-矩阵乘运算的汇编实现, 使函数整体性能提升了 6 倍. 由于在 GEMM KER 版本中, DMA 传输开销仍占有很大比例, 因此, 此时函数的整体性能仅为峰值性能的 5%. 在 RMA KER 版本中, 我们使用基于 RMA 机制的数据共享策略, 大大减少了从核的访存量, 降低了 DMA 传输开销, 使 TRSM 整体性能提升了约 1.6 倍, 为峰值性能的 8%. 由于每次迭代, 非对角线部分的 GEMM 操作依赖对角线部分 TRSM 操作所求出的解, 因此, 基于 RMA 机制的数据共享策略和基于向量化机制的矩阵-矩阵乘运算的汇编实现对函数整体性能的提升有限. 在 TRSM KER 版本中, 加入矩阵方程组求解的汇编实现后, 处于关键路径上的对角线部分 TRSM 操作的性能有了显著提高, 因此函数整体性能进一步加速了约 7.9 倍, 为峰值性能的 63%. 在 TB 版本中, 我们为非对角线部分的 GEMM 操作加入了三缓冲技术, 与图 9 的 GEMM 类似, 使 DMA 访存开销和 RMA 通信开销被计算隐藏, 函数性能得到了约 17% 的提升, 为峰值性能的 74%.

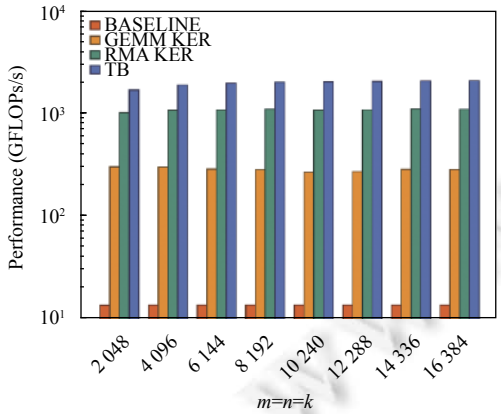


图 9 GEMM 不同优化方法的性能对比

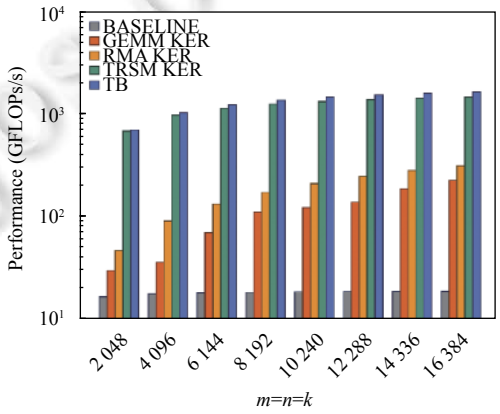


图 10 TRSM 不同优化方法的性能对比

此外, 我们以矩阵均为转置形式的 GEMM 作为实验对象, 验证了汇编实现的矩阵转置操作对 GEMM 的加速效果. 实验结果如图 11 所示. 其中, SCALAR 表示 GEMM 未采用汇编实现的矩阵转置操作的性能, PLUT 表示 GEMM 采用汇编实现的矩阵转置操作的性能. 因为相较于未进行向量化的矩阵转置操作, 单从核的矩阵转置操作加速了约 4 倍, 所以, 相比于 SCALAR, PLUT 的 GEMM 性能有 53%–64% 的提升.

4.2 BLAS 3 级函数整体性能结果

本节将展示所有优化技术用于各个 BLAS 3 级函数时, 它们的整体性能. 在实验中, 我们选取的矩阵规模为 16384, 实验结果如图 12 所示. BLAS 3 级函数的平均性能为 1 779.21 GFLOPs/s, GEMM 的性能可以达到 2 116.04 GFLOPs/s, 相当于单核组浮点计算峰值性能的 92%. SYMM、SYRK、SYR2K、TRMM 的性能略低于 GEMM 的性能, 主要是因为它们包含一些特殊处理, 比如对角线部分的数据需要补全, 这一部分的开销无法通过计算掩盖, 所以整体性能相比 GEMM 略有下降. TRSM 的性能略低于 GEMM 的性能, 主要是因为它在矩阵-矩阵乘法操作的基础上, 还包含回代求解操作, 在求解前, 需要将矩阵的对角元素替换为它的倒数或负倒数, 这一部分的开销无法隐藏, 所以性能相比 GEMM 略有下降.

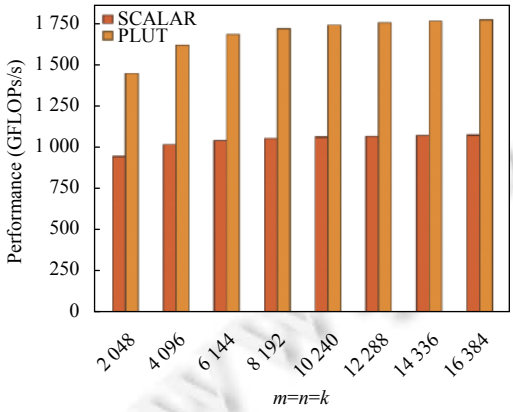


图 11 GEMM 采用汇编转置前后的性能对比图

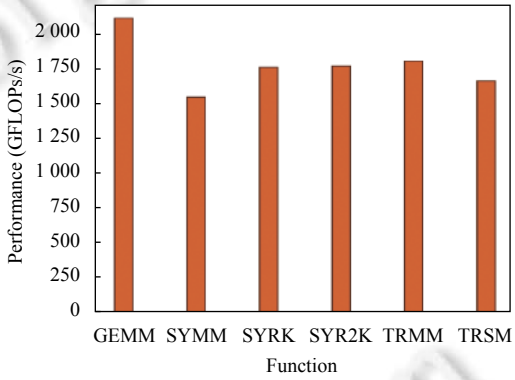


图 12 BLAS 3 级各个函数的性能

4.3 BLAS 3 级函数多核组并行的性能结果

我们在本节将展示 BLAS 3 级函数多核组并行的性能结果. 在实验中, 矩阵规模 (m, n, k) 为 16384, 98304, 16384, 核组数为 6, 实验结果如后文图 13 所示. 在多核组层面, BLAS 3 级函数的平均性能为 10180.24 GFLOPs/s, GEMM 的性能可以达到 12108.02 GFLOPs/s, 相当于多核组浮点计算峰值性能的 88%. 多核组 GEMM 的浮点计算性能比单核组下降 4%. 因为多核组较单核组 DMA 访存带宽下降约 14% (见第 1.2 节), 在三缓冲流水中, 矩阵 A 和 B 在 k 层循环有预取, 预取的开销不能被计算掩盖. 当 DMA 访存带宽下降时, 这部分预取开销所占比例增大, 使得单核组 GEMM 的浮点计算效率降低.

4.4 核组可扩展性分析

本文所做优化基于单个处理器 (包含 6 个核组) 开展, 当核组数为 1, 2, 3, 4, 5, 6 时, 我们使用所有核组内存共享的大共享模式实现了多核组并行. 当核组数大于 6 时, 不同处理器的核组之间无法共享内存, 这种基于分布式存储的 BLAS 函数实现需要使用消息传递等机制实现多进程并行, 不属于本文讨论的范畴. 因此, 本节将展示单个处理器的核组数目对效率的影响.

我们以 GEMM 为实验对象, 展示核组数分别为 1, 2, 3, 4, 5, 6 时, 函数的性能结果. 表 1 和表 2 是核组强/弱可扩展性实验结果, 从中可看出, 函数的效率随着核组数的增加逐渐下降. 当核组数为 6 时, 函数的效率与第 4.3 节的实验结果基本一致. 由此可见, 当单核组分到的矩阵规模足够大时, 除 DMA 访存带宽外, 其余因素对函数的可扩展性以及效率无明显影响.

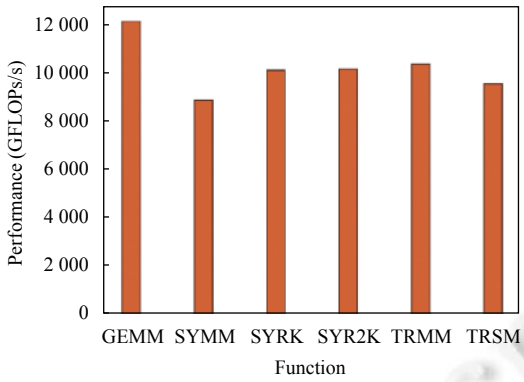


图 13 BLAS 3 级各个函数在整个处理器上的性能

表 1 核组强可扩展性实验结果

核组数	矩阵规模 ($m\times n\times k$)	性能 (GFLOPs/s)	平台峰值性能 (GFLOPs/s)	效率 (%)
1	16384×24576×16384	2 122.19	2 304	92.11
2	16384×24576×16384	4 181.61	4 608	90.75
3	16384×24576×16384	6 156.52	6 912	89.07
4	16384×24576×16384	8 136.81	9 216	88.29
5	16384×24576×16384	10 128.38	11 520	87.92
6	16384×24576×16384	11 928.73	13 824	86.29

表 2 核组弱可扩展性实验结果

核组数	矩阵规模 ($m\times n\times k$)	性能 (GFLOPs/s)	平台峰值性能 (GFLOPs/s)	效率 (%)
1	16384×16384×16384	2 116.04	2 304	91.84
2	16384×32768×16384	4 203.97	4 608	91.23
3	16384×49152×16384	6 247.76	6 912	90.39
4	16384×65536×16384	8 227.95	9 216	89.28
5	16384×81920×16384	10 220.31	11 520	88.72
6	16384×98304×16384	12 108.02	13 824	87.59

5 结 论

本文基于 SW26010-Pro 众核计算平台, 提出了若干 BLAS 3 级函数的并行算法和优化方法. 本文设计了一种多级分块算法, 以挖掘矩阵运算的并行性. 在此基础上, 基于 RMA 机制设计了数据共享策略, 提高了从核间的数据传输效率. 特别地, 本文采用了多种优化方法对算法进行深度优化. 本文采用了三缓冲技术隐藏了 DMA 访存开销和 RMA 通信开销, 并通过手工汇编优化提高了 BLAS 3 级函数的浮点计算效率. 实验结果说明, 本文提出的优化方案为 BLAS 3 级函数带来了明显的性能提升, 其中, 单核组 GEMM 的浮点计算性能达到峰值性能的 92%, 多核组 GEMM 的浮点计算性能达到峰值性能的 88%.

在不同应用中, GEMM 的输入矩阵规模或形状通常会有所不同. 例如, 人工神经网络 (ANN)[22]和高度优化的 K-means[23]使用 GEMM 作为其核心计算, 输入矩阵的形状通常是非规整的 (non-regular shaped)[24–26]. 已有的 GEMM 优化工作主要涉及形状规整 (regular shaped) 的输入矩阵[27–29], 对于非规整的输入矩阵, 其性能较低. 本文的优化工作也主要面向较大规模矩阵, 对于非规整矩阵, 其性能较低. 因此, 对非规整输入矩阵的 GEMM 的优化是下一步工作中需要深入探索并解决的一个重要问题.

References:

[1] Blackford LS, Demmel J, Dongarra J, Duff I, Hammarling S, Henry G, Heroux M, Kaufman L, Lumsdaine A, Petitet A, Pozo R, Remington K, Whaley RC. An updated set of basic linear algebra subprograms (BLAS). ACM Trans. on Mathematical Software, 2002, 28(2): 135–151. [doi: 10.1145/567806.567807]

[2] Van de Geijn R, Goto K. BLAS (basic linear algebra subprograms). In: Padua D, ed. Encyclopedia of Parallel Computing. Boston: Springer, 2011. 157–164. [doi: 10.1007/978-0-387-09766-4_84]

[3] Hong CE, McMillin BM. Fault-tolerant parallel matrix multiplication with one iteration fault detection latency. In: Proc. of the 15th Annual Int'l Computer Software and Applications Conf. Tokyo: IEEE, 1991. 665–672. [doi: 10.1109/CMPSAC.1991.170258]

[4] Plancher B, Brumar CD, Brumar I, Pentecost L, Rama S, Brooks D. Application of approximate matrix multiplication to neural networks and distributed SLAM. In: Proc. of the 2019 IEEE High Performance Extreme Computing Conf. Waltham: IEEE, 2019. 1–7. [doi: 10.1109/HPEC.2019.8916468]

[5] Georgescu IM, Ashhab S, Nori F. Quantum simulation. Physics, 2014, 86(1): 153–185. [doi: 10.1103/RevModPhys.86.153]

[6] Shields DS. Prospecting for oil. Gastronomica, 2010, 10(4): 25–34. [doi: 10.1525/gfc.2010.10.4.25]

[7] Kågström BO, Ling P, van Loan C. GEMM-based level 3 BLAS: High-performance model implementations and performance evaluation benchmark. ACM Trans. on Mathematical Software, 1998, 24(3): 268–302. [doi: 10.1145/292395.292412]

[8] INTEL. Intel®-optimized math library for numerical computing. 2021. https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onemkl.html

[9] AMD. AMD optimizing CPU libraries. 2021. https://developer.amd.com/amd-aocl/

[10] NVIDIA. Basic linear algebra on NVIDIA GPUs. 2021. https://developer.nvidia.com/cublas

[11] Goto K, Van De Geijn R. High-performance implementation of the level-3 BLAS. ACM Trans. on Mathematical Software, 2008, 35(1): 4. [doi: 10.1145/1377603.1377607]

[12] Clint Whaley R, Petitet A, Dongarra JJ. Automated empirical optimizations of software and the atlas project. Parallel Computing, 2001, 27(1–2): 3–35. [doi: 10.1016/S0167-8191(00)00087-9]

[13] Zhang XY, Kroeker M. OpenBLAS. 2023. http://www.openblas.net/

[14] Jiang LJ, Yang C, Ao YL, Yin WW, Ma WJ, Sun Q, Liu FF, Lin RF, Zhang P. Towards highly efficient DGEMM on the emerging SW26010 many-core processor. In: Proc. of the 46th Int'l Conf. on Parallel Processing. Bristol: IEEE, 2017. 422–431. [doi: 10.1109/ICPP.2017.51]

[15] Goto K, van de Geijn R A. Anatomy of high-performance matrix multiplication. ACM Trans. on Mathematical Software, 2008, 34(3): 12. [doi: 10.1145/1356052.1356053]

[16] Wang Q, Zhang XY, Zhang YQ, Yi Q. AUGEM: Automatically generate high performance dense linear algebra kernels on x86 CPUs. In: Proc. of the 2013 Int'l Conf. on High Performance Computing, Networking, Storage and Analysis. Denver: ACM, 2013. 25. [doi: 10.1145/2503210.2503219]

[17] Zhang XY, Wang Q, Zhang YQ. Model-driven level 3 BLAS performance optimization on Loongson 3A processor. In: Proc. of the 18th Int'l Conf. on Parallel and Distributed Systems. Singapore: IEEE, 2012. 684–691. [doi: 10.1109/ICPADS.2012.97]

[18] Nath R, Tomov S, Dongarra J. BLAS for GPUs. In: Scientific Computing with Multicore and Accelerators. Boca Raton: CRC Press, 2010. 57–80.

[19] Wang LN, Wu W, Xiao JX, Yang Y. BLASX: A high performance level-3 BLAS library for heterogeneous multi-GPU computing. arXiv:1510.05041, 2015.

[20] Igual FD, Quintana-Ortí G, Van De Geijn RA. Level-3 BLAS on a GPU: Picking the low hanging fruit. AIP Conf. Proc., 2012, 1504(1): 1109–1112. [doi: 10.1063/1.4772121]

[21] Liu H, Liu FF, Zhang P, Yang C, Jiang LJ. Optimization of BLAS level 3 functions on SW1600. Computer Systems & Applications, 2016, 25(12): 234–239 (in Chinese with English abstract). [doi: 10.15888/j.cnki.csa.005456]

[22] Hopfield JJ. Artificial neural networks. IEEE Circuits and Devices Magazine, 1988, 4(5): 3–10. [doi: 10.1109/101.8118]

[23] Dhillon IS, Guan YQ, Kulis B. Kernel K-means: Spectral clustering and normalized cuts. In: Proc. of the 10th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. Seattle: ACM, 2004. 551–556. [doi: 10.1145/1014052.1014118]

[24] Chen JY, Xiong N, Liang X, Tao DW, Li SH, Ouyang KM, Zhao K, DeBardeleben N, Guan Q, Chen ZZ. TSM2: Optimizing tall-and-skinny matrix-matrix multiplication on GPUs. In: Proc. of the 2019 ACM Int'l Conf. on Supercomputing. Phoenix: ACM, 2019. 106–116. [doi: 10.1145/3330345.3330355]

[25] Jhurani C, Mullowney P. A GEMM interface and implementation on NVIDIA GPUs for multiple small matrices. Journal of Parallel and Distributed Computing, 2015, 75: 133–140. [doi: 10.1016/j.jpdc.2014.09.003]

[26] Ernst D, Hager G, Thies J, Wellein G. Performance engineering for real and complex tall & skinny matrix multiplication kernels on GPUs. The Int'l Journal of High Performance Computing Applications, 2021, 35(1): 5–19. [doi: 10.1177/1094342020965661]

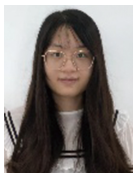
[27] Goto K, Van De Geijn R. On reducing TLB misses in matrix multiplication. 2002. http://users.umiacs.umd.edu/~ramani/cmsc662/Goto_vdGeijn.pdf

[28] Nath R, Tomov S, Dongarra J. An improved magma GEMM for Fermi graphics processing units. The Int'l Journal of High Performance Computing Applications, 2010, 24(4): 511–515. [doi: 10.1177/1094342010385729]

[29] Lancee B, Birkelund G, Coenders M, Di Stasio V, Fernandez Reino M, Heath A, Koopmans R, Larsen E, Polavieja JG, Ramos M, Soiné H, Thijssen L, Veit S, Yemane R. The GEMM study: A cross-national harmonized field experiment on labour market discrimination. 2019. https://gemm2020.eu/wp-content/uploads/2019/02/GEMM-WP3-technical-report.pdf

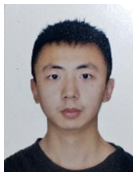
附中文参考文献:

[21] 刘昊, 刘芳芳, 张鹏, 杨超, 蒋丽娟. 基于申威1600的3级BLAS GEMM函数优化. 计算机系统应用, 2016, 25(12): 234–239. [doi: 10.15888/j.cnki.csa.005456]



胡怡(1995－), 女, 博士生, 主要研究领域为高性能计算, 异构并行, BLAS 库, 稠密矩阵相关的算法研究.

刘芳芳(1982－), 女, 正高级工程师, CCF 专业会员, 主要研究领域为高性能扩展数学库, 超级计算机评测软件.



陈道琨(1994－), 男, 博士生, 主要研究领域为高性能计算, 异构并行, 稀疏矩阵相关的算法研究.

宋超博(1994－), 男, 本科, 主要研究领域为高性能计算.

杨超(1979－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为高性能计算, 科学与工程计算.

孙强(1993－), 男, 助理工程师, 主要研究领域为网络空间安全, 高性能计算.

马文静(1981－), 女, 副研究员, CCF 专业会员, 主要研究领域为高性能计算, 代码生成与优化.

史俊达(1997－), 男, 助理工程师, 主要研究领域为高性能计算.