软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2023,34(9):4421−4436 [doi: 10.13328/j.cnki.jos.006527]

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

面向 SW26010-Pro 的 1、2 级 BLAS 函数众核并行优化技术*

胡 怡[1,2], 陈道琨[1,2], 杨 超[3], 刘芳芳[1,2], 马文静[1,2], 尹万旺[4], 袁欣辉[4], 林蓉芬[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) 是高性能扩展数学库的一个重要模块, 广泛应用于科学与工程计算领域. BLAS 1 级提供向量-向量运算, BLAS 2 级提供矩阵-向量运算. 针对国产 SW26010-Pro 众核处理器设计并实现了高性能 BLAS 1、2 级函数. 基于 RMA 通信机制设计了从核归约策略, 提升了 BLAS 1、2 级若干函数的归约效率. 针对 TRSV、TPSV 等存在数据依赖关系的函数, 提出了一套高效并行算法, 该算法通过点对点同步维持数据依赖关系, 设计了适用于三角矩阵的高效任务映射机制, 有效减少了从核点对点同步的次数, 提高了函数的执行效率. 通过自适应优化、向量压缩、数据复用等技术, 进一步提升了 BLAS 1、2 级函数的访存带宽利用率. 实验结果显示, BLAS 1 级函数的访存带宽利用率最高可达 95%, 平均可达 90% 以上, BLAS 2 级函数的访存带宽利用率最高可达 98%, 平均可达 80% 以上. 与广泛使用的开源数学库 GotoBLAS 相比, BLAS 1、2 级函数分别取得了平均 18.78 倍和 25.96 倍的加速效果. LU 分解、QR 分解以及对称特征值问题通过调用所提出的高性能 BLAS 1、2 级函数取得了平均 10.99 倍的加速效果.

关键词: BLAS 1 级; BLAS 2 级; 访存带宽; SW26010-Pro 众核处理器; RMA 通信; 点对点同步; 自适应优化

中图法分类号: TP303

中文引用格式: 胡怡, 陈道琨, 杨超, 刘芳芳, 马文静, 尹万旺, 袁欣辉, 林蓉芬. 面向SW26010-Pro的1、2级BLAS函数众核并行优化技术. 软件学报, 2023, 34(9): 4421–4436. http://www.jos.org.cn/1000-9825/6527.htm

英文引用格式: Hu Y, Chen DK, Yang C, Liu FF, Ma WJ, Yin WW, Yuan XH, Lin RF. Many-core Optimization of Level 1 and Level 2 BLAS Routines on SW26010-Pro. Ruan Jian Xue Bao/Journal of Software, 2023, 34(9): 4421–4436 (in Chinese). http://www.jos.org.cn/1000-9825/6527.htm

Many-core Optimization of Level 1 and Level 2 BLAS Routines on SW26010-Pro

HU Yi[1,2], CHEN Dao-Kun[1,2], YANG Chao[3], LIU Fang-Fang[1,2], MA Wen-Jing[1,2], YIN Wan-Wang[4], YUAN Xin-Hui[4], LIN Rong-Fen[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: BLAS (basic linear algebra subprograms) is an important module of the high-performance extended math library, which is widely used in the field of scientific and engineering computing. Level 1 BLAS provides vector-vector operation, Level 2 BLAS provides matrix-vector operation. This study designs and implements highly optimized Level 1 and Level 2 BLAS routines for SW26010-Pro, a domestic many-core processor. A reduction strategy among CPEs is designed based on the RMA communication mechanism, which

* 基金项目: 国家重点研发计划 (2020YFB0204601)
收稿时间: 2021-07-02; 修改时间: 2021-09-22; 采用时间: 2021-11-05; jos 在线出版时间: 2022-11-30
CNKI 网络首发时间: 2022-12-01

improves the reduction efficiency of many Level 1 and Level 2 BLAS routines. For TRSV and TPSV and other routines that have data dependencies, a series of efficient parallelization algorithms are proposed. The algorithm maintains data dependencies through point-to-point synchronization and designs an efficient task mapping mechanism that is suitable for triangular matrices, which reduces the number of point-to-point synchronizations effectively, and improves the execution efficiency. In this study, adaptive optimization, vector compression, data multiplexing, and other technologies have further improved the memory access bandwidth utilization of Level 1 and Level 2 BLAS routines. The experimental results show that the memory access bandwidth utilization rate of the Level 1 BLAS routines can reach as high as 95%, with an average bandwidth of more than 90%. The memory access bandwidth utilization rate of Level 2 BLAS routines can reach 98%, with an average bandwidth of more than 80%. Compared with the widely used open-source linear algebra library GotoBLAS, the proposed implementation of Level 1 and Level 2 BLAS routines achieved an average speedup of 18.78 times and 25.96 times. With the optimized Level 1 and Level 2 BLAS routines, LQ decomposition, QR decomposition, and eigenvalue problems achieved an average speedup of 10.99 times.

Key words: level 1 BLAS; level 2 BLAS; memory access bandwidth; Sunway 26010-Pro many-core processor; RMA communication; point-to-point synchronization; adaptive optimization

BLAS (basic linear algebra subprograms) 是最基本、最重要的底层数学库之一, 也是高性能扩展数学库中重要的模块之一[1], 在科学计算领域、机器学习以及人工智能领域应用广泛[2−4]. 许多大规模科学计算应用如天体物理[5]、量子化学[6]、大气动力学[7,8]和材料科学[9]等, 其底层均依赖于高度优化的 BLAS 函数. 因此, 设计高性能 BLAS 数学库对提高应用的计算速度有重要作用. 在一个标准的 BLAS 库中, 以向量-向量运算函数为代表的 BLAS 1 级函数和以矩阵-向量运算函数为代表的 BLAS 2 级函数会被几乎所有与稠密线性代数运算有关的应用程序或软件 (如 LAPACK[10], ScaLAPACK[11]等) 反复多次调用, 在诸如矩阵分解[12−14]、矩阵特征值求解[15,16]等领域有广泛的应用, 其高性能实现技术具有重要的实用价值和研究意义. 不同于以矩阵-矩阵运算函数为代表的 BLAS 3 级函数所具有计算密集型特征, BLAS 1、2 级函数主要是访存密集型任务, 其访存开销无法通过计算隐藏, 因此, 为了实现高性能 BLAS 1、2 级函数, 必须设计合理的并行策略, 充分挖掘与访存有关的硬件特性, 从而释放处理器的性能.

研究人员已在 CPU、GPU 等通用高性能计算平台上开展了与 BLAS 1、2 级函数高性能并行优化技术相关的大量研究工作[17−19]. 随着国产处理器的飞速发展以及国产超级计算机的研制和应用水平提升, 开展高性能 BLAS 1、2 级函数库的研制和优化迫在眉睫. 由于硬件工艺方面的限制, 国产处理器的硬件架构、指令集、存储层次等一般与通用 CPU、GPU 存在较大差异, 一些常用的 BLAS 1、2 级函数优化手段并不适用. 因此, 针对国产处理器特有的体系结构特征, 开展以 BLAS 1、2 级函数等为代表的高性能数学函数库的性能优化技术研究具有重要意义. 近期研制完成的神威新一代超级计算机配备了我国自主研发的主从核异构的 SW26010-Pro 众核处理器. 相比于神威太湖之光所配备的 SW26010 处理器, SW26010-Pro 在单核组运算性能方面提升幅度达到 3.1 倍[20], 但是在访存带宽方面仅提升 1.8 倍[21], 总体访存性能与计算性能之间的差距进一步拉大, 这给具有访存密集型特征的 BLAS 1、2 级函数的高性能实现带来了巨大挑战. 不仅如此, SW26010-Pro 众核处理器在硬件设计上做了重大调整, 比如, 把寄存器通信机制调整为 RMA 通信, 还加入了点对点同步机制等, 对向量化宽度和 LDM 空间大小也进行了修改. 因此, 为了最大程度挖掘 SW26010-Pro 的硬件性能, 需要针对 SW26010-Pro 众核处理器为 BLAS 1、2 级函数设计新的并行策略和优化方案.

本文针对 SW26010-Pro 众核处理器的体系结构特征以及硬件特性, 设计并实现了整套高性能 BLAS 1、2 级函数, 从任务划分、LDM 空间划分以及减少冗余访存等方面展开工作. 具体而言, 本文基于 RMA 通信机制设计了从核归约策略, 提升了 BLAS 1、2 级若干函数的归约效率; 针对 TRSV、TPSV 等存在数据依赖关系的函数, 本文根据 SW26010-Pro 众核处理器的新特性提出了一套高效并行算法, 该算法通过点对点同步维持数据依赖关系, 设计了适用于三角矩阵的高效任务映射机制, 有效减少了从核点对点同步的次数, 提高了函数的执行效率; 此外, 本文通过自适应优化、向量压缩、数据复用等技术, 进一步提升了 BLAS 1、2 级函数的访存带宽利用率. 本文实现的 BLAS 1 级函数的平均访存带宽利用率达到了峰值带宽的 90% 以上, BLAS 2 级函数的平均访存带宽利用率达到了峰值带宽的 80% 以上. 相比于 GotoBLAS 0.2.6[22], BLAS 1、2 级函数分别取得了平均 18.78 倍和 25.96 倍的加速效果, LU 分解[13]、QR 分解[14]以及对称特征值问题[15,16]通过调用本文实现的高性能 BLAS 1、2 级函数取得了平均 10.99 倍的加速效果.

1 背景介绍

1.1 相关工作

在 BLAS 1、2 级函数的并行算法与性能优化方面, 研究机构及处理器厂商已经开展了大量研究工作. 如美国德克萨斯州大学超级计算中心针对 Power、Alpha 等主流处理器开发的 GotoBLAS 库[23]; Intel、AMD 针对 x86 架构研发的 MKL[24]、AOCL[25]; Nvidia 针对众核 GPGPU 研发的 cuBLAS 数学库[26]. 此外, 还有通过分析平台性能参数进行自动调优的 ATLAS 数学库[27]. 随着异构众核并行计算的兴起, 国内外许多科研机构也对 BLAS 1、2 级函数进行了深入的优化技术研究. 如中国科学院计算技术研究所针对类 Beowulf 机群系统实现的高性能 BLAS 数学库[28]; 中国科学技术大学基于龙芯 2F 体系结构优化的 BLAS 数学库[29]; 中国科学院软件研究所针对国产申威 1600 平台和国产申威 26010 平台实现的 xMath 数学库[30,31]. 以上这些数学库中都包含了 BLAS 1、2 级函数, 并且为多个应用提供了性能支撑. 此外, 国内外许多学者针对 BLAS 1、2 级的某些具体函数进行了高度优化. 如 Yin 等人在 Nvidia GPU 上利用寄存器分块方法实现了并行 GEMV[32]; Xu 等人在 Nvidia GPU 上实现了一个针对 GEMV 的性能调优框架, 针对 GEMV 的输入规模选择最优的算法[33]; Nath 等人在 Nvidia GPU 上提出了实现 SYMV 的两种并行算法[34]; Li 等人面向多核龙芯 3A 实现了二级 BLAS 函数库[35]; Chohra 等人实现了高度优化的 BLAS 1 级函数[36]; Imamura 等人在 Nvidia Fermi 架构 GPU 上利用原子运算实现了高性能的 SYMV[37]; Wang 等人在多核 CPU 上实现了并行 BLAS 1、2、3 级函数[38]. 本文在充分借鉴上述工作基础上, 独立发展了一套面向 SW26010-Pro 众核处理器的 BLAS 1、2 级函数并行实现与性能优化技术.

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 个核组和系统接口组成. 由于高性能扩展数学库功能方面的需要, 本文围绕 SW26010-Pro 处理器的单一核组展开工作. 每个核组主要包括 1 个主核 (MPE) 和 1 个从核 (CPE) 阵列, 主核采用 64 位 RISC 结构通用处理单元. 从核阵列由 64 个从核构成, 以 8×8 列阵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, 每个线程与对应的从核进行绑定. 为方便描述, 我们用 C_i 表示 i 号从核, 用 T_i 表示运行在 C_i 上的 i 号线程.

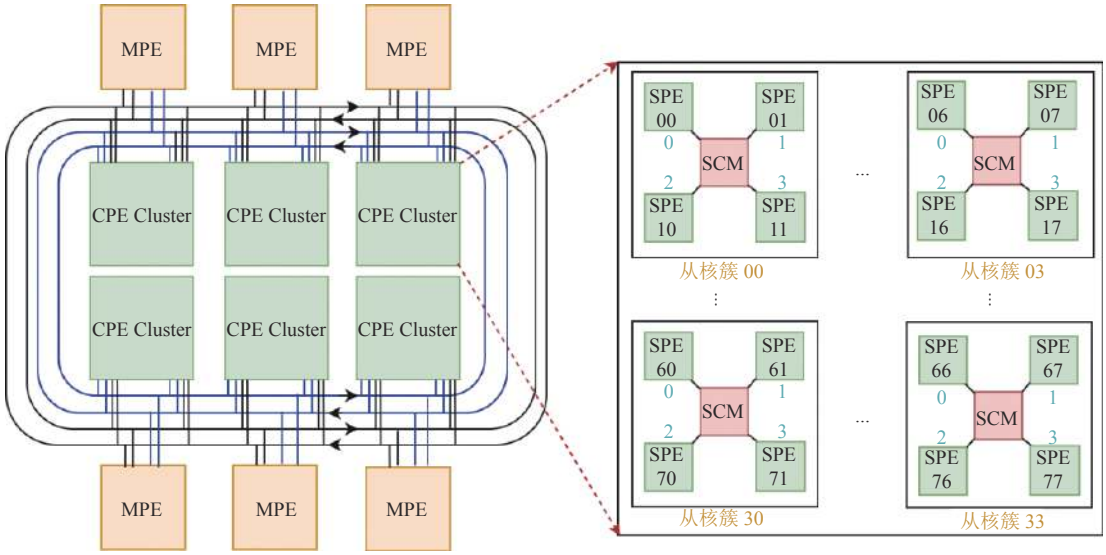


图 1 SW26010-Pro 架构以及从核阵列结构逻辑示意图

SW26010-Pro 众核处理器一个核组内的存储空间由主核的主存空间和每个从核的 256 KB 私有 LDM (local data memory, 局部数据存储器) 空间组成. 主从核间的数据交换除直接读写内存外, 还可通过专用 DMA (direct

memory access, 直接内存访问) 通道高速传输成块的数据. DMA 支持带跨步和不带跨步两种形式, 单从核的 DMA 访存带宽取决于跨步向量块 (*bsize*) 的大小. 当 DMA 操作不带跨步时, *bsize* 的大小为数据量的大小. 多从核的 DMA 访存带宽正比于单从核的 DMA 访存带宽, 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. 从核阵列内部使用 RMA (remote memory access, 远程内存访问) 通道通信. RMA 包含多种传输类型, 其中常用的是单从核模式与广播模式, 实现单个从核与单个/多个从核的数据交换. DMA 和 RMA 操作的数据地址必须 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、从核分别支持 256 位、512 位的向量化处理长度. 此外, 从核支持 IEEE754 标准的半精度、单精度和双精度浮点数据类型. 单个从核阵列的浮点运算峰值性能为 16 flop/cycle×2.25 GHz×64=2304 GFLOPs/s.

1.3 BLAS 1 级和 BLAS 2 级函数简介

BLAS 1 级函数实现标量-向量、向量-向量运算, 共包含 17 个函数, 访存复杂度与计算复杂度均为 $\mathrm{O}(n)$. 本文以函数 AXPY (标量向量乘) 为例来说明 BLAS 1 级函数的计算形式, 该函数包含实数单/双精度、复数单/双精度 4 种数据类型, 其计算过程见公式 (1):

$$y=\alpha\times x+y \tag{1}$$

其中, α 为标量, x, y 为 n 维向量.

BLAS 2 级函数实现矩阵-向量运算, 共包含 25 个函数, 访存复杂度与计算复杂度均为 $\mathrm{O}\left(n^2\right)$. BLAS 2 级函数主要包含以下几种矩阵类型: 普通矩阵、带状矩阵、Packed 矩阵、Hermitian 矩阵、三角矩阵、对称矩阵. 这里以 BLAS 2 级中具有代表性的函数 GEMV (普通矩阵向量乘) 为例来说明 BLAS 2 级函数的计算形式, 该函数包含实数单/双精度、复数单/双精度 4 种数据类型, 其计算过程见公式 (2):

$$y=\alpha\times A\times x+\beta\times y \tag{2}$$

其中 α、β 为标量, A 为 $m\times n$ 的矩阵, 可以是转置或非转置形式, x、y 分别为 n 维、m 维向量.

2 并行实现

2.1 BLAS 1 级函数

如上文所述, BLAS 1 级函数仅处理向量这一种对象, 它的运算是向量对应元素之间的独立计算, 所以通过对向量进行分段处理可实现并行化. 例如, AXPY 的算法如算法 1 所示, 其中 t 是参与本次运算的线程数量, *id* 表示从核的线程号, B 是分块参数, 取决于 LDM 的空间大小. 算法 1 的主从核数据交换通过阻塞式的 DMA 操作实现, 第 6 行的计算可以利用对应的 SIMD 向量化指令实现加速. 由于多个从核同时进行 DMA 读写操作会出现硬件资源争用的现象, 造成带宽下降, 所以我们在 DMA 读和 DMA 写之间引入线程同步避免资源争用 (算法 1 的第 4 行). DOT、IAMAX、IAMIN 等函数的输出结果还需要对向量 y 进行归约, 每个从核先累加自己的结果形成部分和, 经主核归约得到最终结果.

算法 1. *AXPY* 的并行算法, $AXPY(n,\alpha,x[n],incx,y[n],incy)$.

1. $len=n/t$
2. **for** $i=id\times len$ **to** $len-B$ **by** B **do**
3. 读取向量段 x_i, y_i 到 *id* 号从核的 LDM
4. 从核阵列同步
5. 并行计算向量段每个分量:
6. $y_i[j]+=\alpha*x_i[j]$
7. *id* 号从核写回向量段 y_i 到主内存
8. **end for**

2.2 BLAS 2 级函数

BLAS 2 级函数的操作对象是矩阵和向量, 如 GEMV、SYMV、GER、TRMV 等函数是对矩阵-向量对应元素之间的独立计算, 所以通过对矩阵进行分块处理可实现并行化. TRSV、TPSV 等函数的解向量之间存在依赖, 在并行算法的设计中需要考虑依赖关系, 所以在求解过程中需要引入线程归约、线程同步以实现并行化. 在下文中我们详细介绍了 GEMV、SYMV 和 TRSV 的并行算法, 这 3 个函数代表了 BLAS 2 级函数的计算特点, 其他 BLAS 2 级函数的矩阵存储特点和计算流程虽与之不同, 但实现方式大同小异.

2.2.1 矩阵向量乘

对于普通稠密矩阵-向量乘如 GEMV、GER 等函数, 矩阵的形式比较规整, 我们使用静态任务划分的方法对其进行分块处理, 如图 2. 其中 t 是参与本次任务划分的线程数量, $nrows$ 是每个从核获得的任务量. 对于三角矩阵或对称矩阵如 TRMV、SYMV 等函数, 为了保证从核负载均衡, 我们使用动态任务划分的方法将矩阵分成若干块, 如图 3. 其中 $bsize$ 是 DMA 操作的跨步向量块的大小, 我们对矩阵按照 $bsize$ 进行块划分, 将每个块的计算视为一个子任务, 形成任务池, 每个从核一次执行一个子任务. 当每个从核执行完当前子任务后, 利用原子加 1 操作, 及时从任务池中获取下一个子任务, 这样就避免了从核的空闲, 保证了各从核的负载均衡. 在下文中我们以 GEMV 和 SYMV 为例详细介绍了矩阵向量乘的并行算法.

GEMV 主要实现: $y=\alpha Ax+\beta y$, 其中 A 是大小为 $m\times n$ 的普通矩阵, x、y 是向量, α、β 是标量参数. 本文以矩阵 A 非转置的情况为例来说明 GEMV 的算法. 图 4 是 GEMV 的分块示意图, 采用了上文所述的静态任务划分方式. 矩阵 A 在逻辑上被划分成若干小矩阵, 每个线程处理一个行块, 采用这种划分方式目的是为了复用向量 x. 算法 2 给出了 GEMV 的实现过程, 其中 t 是参与本次运算的线程数量, id 表示从核的线程号. 算法 2 的主从核数据交换通过阻塞式的 DMA 操作实现, 小矩阵的规模是 $nrows\times bsize$, $bsize$ 是 DMA 操作的跨步向量块的大小. 若 DMA 的参数 $incx$ 和 $incy$ 大于 1, 则向量将进行非连续的 DMA 操作, 降低了 DMA 的带宽利用率. 因此, 在算法 2 之前, 本文对向量 x 和向量 y 进行了压缩, 如算法 3 所示. 其中, $incx$ 是向量的增量, B 是分块参数, 取决于 LDM 的空间大小. 算法 2 的计算部分可以利用 SIMD 向量化指令实现加速.

算法 2. $GEMV$ 的并行算法 $GEMV(m,n,\alpha,a[m\times n],lda,x[n],incx,\beta,y[n],incy)$.

1. $nrows=\min(m/t,bsize)$
2. **for** i=(id×$nrows$) **to** ((id+1)×$nrows$) **by** $bsize$ **do**
3. 读取向量段 y_i 到 id 号从核的 LDM, $y_i[r]*=\beta$
4. **for** $j=0$ **to** $n-bsize$ **by** $bsize$ **do**
5. 读取向量段 x_j, 矩阵块 a_{ij} 到 id 号从核的 LDM
6. 并行计算矩阵块和向量段的每个分量:
7. $y_i[r]+=\alpha\times a_{ij}[r][c]\times x_j[c]$
8. **end for**
9. id 号从核写回向量段 y_i 到主内存
10. **end for**

算法 3. 向量压缩技术 $Vector\ Compression(n,x[n],incx)$.

1. **for** i=0 **to** $n-B$ **by** B **do**
2. 从核以阻塞方式发起 DMA 读操作, 将数据从主内存读到 LDM, 此时 DMA 的参数 $incx$ 大于 1
3. 从核以阻塞方式发起 DMA 写操作, 将数据从 LDM 写回主内存, 将 DMA 的参数 $incx$ 设为 1
4. **end for**

SYMV 函数是对称矩阵向量乘, 本文以下三角对称矩阵为例来说明它的算法. 图 5 是 SYMV 的分块示意图, 采用了上文所述的动态任务划分方式. 矩阵 A 在逻辑上被划分成若干小矩阵, 每个线程处理一个列块, 采用这种划

分方式的目的是为了复用矩阵 A 对称部分的元素. 与 GEMV 相比, 矩阵的对称性可以减少从核对主存的访问, 但会引起多个从核同时访问向量的冲突. 因此, 为了避免从核对向量的竞读竞写, 在 GEMV 的基础上, SYMV 的算法还需要加入锁机制. 具体做法是以小方阵为单位, 将向量 y 分成若干段对其进行分段加锁, 利用多个锁保护向量 y 中每段数据的读取与写回. 由于 SYMV 和 GEMV 的算法相似, 故 SYMV 的算法在此不再赘述.

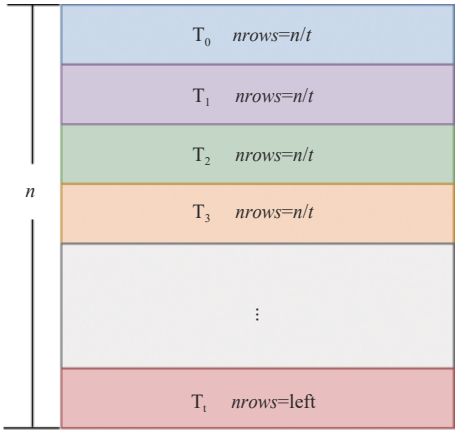


图 2 静态任务划分示意图

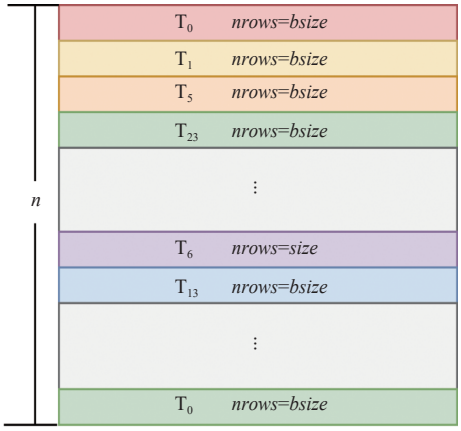


图 3 动态任务划分示意图

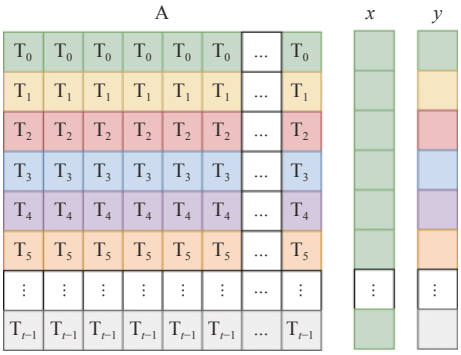


图 4 GEMV 分块示意图

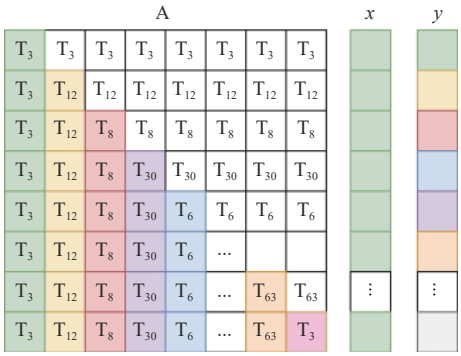


图 5 SYMV 分块示意图

2.2.2 线性方程组求解

与 GEMV 等其他二级函数相比有所不同, 线性方程组求解如 TRSV、TPSV 等函数存在数据依赖关系, 需要为其设计对应的任务划分方式和线程通信机制. 在下文中我们以 TRSV 为例详细介绍了线性方程组求解的并行算法.

$$x_i \leftarrow A_{ii}^{-1} \times \left(b_i - \sum_{0 \leqslant j < i} A_{ij} \times x_j \right) \tag{3}$$

TRSV 是稠密线性方程组求解, 主要求解如下等式: $A \times x = b$. 其中 x 是未知向量, b 是已知向量, A 是大小为 $n \times n$ 的三角矩阵. 本文以下三角矩阵为例进行介绍. 当矩阵 A 的对角元素不为零时, TRSV 可用公式 (3) 表示, 解向量 x 的每个元素通过回代进行求解. 我们将矩阵 A 进行类似 GEMV 的分块, 目的是为了复用向量 x. 在这种分块模式下, 公式 (3) 的 A_{ij} 表示矩阵 A 的第 (i, j) 个分块, x_i 和 b_i 分别表示向量 x 和向量 b 的第 (i) 个分段, x_i 的求解依赖于已求解的 $x_0, x_1, \ldots, x_{(i-1)}$. 根据公式 (3), 本文将 TRSV 分解成矩阵-向量乘和求和两种操作, 在我们的算法中, 前者由 1 至 63 号从核完成, 后者由 0 号从核完成. 因此, 我们将 A_{ii} 映射给 0 号从核, A_{ij} $(0 \leqslant j < i)$ 依次映射给 $((j\%63)+1)$ 号从核. 图 6 是矩阵 A 的分块示意图, 小矩阵的规模是 $bsize \times bsize$, $bsize$ 是 DMA 操作的跨步向量块的大小. 算法 4 给出了 TRSV 的实现过程, 1 至 63 号从核先累加自己的矩阵-向量乘结果, 再将结果归约至 0 号从核, 由 0 号从核计算出解向量 x. 其中第 8、12 行的计算可以利用对应的 SIMD 指令实现加速, 算法中的主从核数据

交换通过阻塞式的 DMA 操作实现.

算法 4. TRSV 的并行算法, $Trsv(n,a[n][n],lda,x[n],incx)$.

1. **for** i=0 **to** $n-1$ **by** *bsize* **do**
2. $total=\lceil i/bsize \rceil$
3. 除 0 号从核外的其他从核进行以下操作 {
4. **for** $index=id-1$ **to** $total-1$ **by** $(t-1)$ **do**
5. $j=index\times bsize$
6. 判断当前是否需要与 0 号从核同步, 如果是, 同步; 如果不是, 直接进行下面的操作
7. 读取向量段 x_j, 矩阵块 a_{ij} 到 *id* 号从核的 LDM
8. 并行计算矩阵块和向量段的每个分量: $y_i[r]+=a_{ij}[r][c]\times x_j[c]$
9. **end for** }
10. 0 号从核读取向量段 x_i, 矩阵块 a_{ii} 到自己的 LDM
11. 从核阵列归约
12. 0 号从核求解向量段 x_i, 并将其写回到主内存
13. 0 号从核与 (*total*<63?(*total*+1):((*total*+1)%63)=0?63:(*total*+1)%63) 号从核同步
14. **end for**

0 号从核在求解完当前向量段后, 将结果写回主存 (算法 4 的第 12 行). 为保证其他从核在进行后面的矩阵-向量乘操作时从主存中取到的是被更新后的向量, 从核间需要通讯. 如图 7, 负责小矩阵 $A_{(i+1)i}$ 的从核 ((*i*%63)+1) 需要等待 0 号从核求解完 x_i 才能发起对 x_i 的 DMA 操作. 在我们的算法中, 0 号从核求解完 x_i 后, 会对 ((*i*%63)+1) 号从核发起一次点对点同步请求 (算法 4 的第 13 行), ((*i*%63)+1) 号从核发起对 x_i 的 DMA 操作前会响应这个同步请求 (算法 4 的第 6 行). 类似的过程可以采用信号量实现. 算法中 0 号从核每次只和一个从核同步, 且知道与之同步的从核号. 因此, 信号量作为一种单向同步机制优势不明显, 不能简化程序的设计. 并且在该平台上信号量通过共享内存实现, 从核使用信号量来通信以协调各自活动可能需要频繁访问对方的 LDM 空间, 访问一次约 50 拍, 而从核点对点同步一次约 30 拍. 因此, 点对点同步相较而言是一个比较合理的方法. 在当前小矩阵-从核的映射模式下, 0 号从核每次只需和一个从核同步. 如果采用 GEMV 的映射方法, 从核单点同步的次数会大大增加, 进而增加程序的开销.

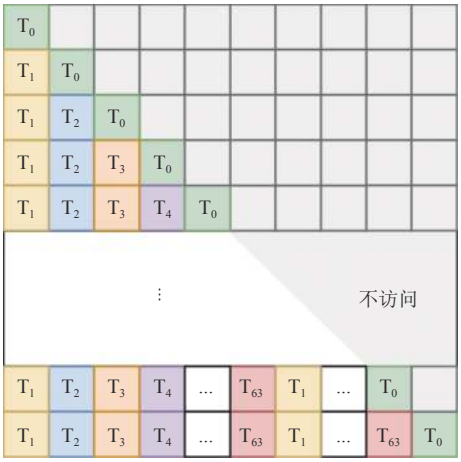


图 6 下三角矩阵分块示意图

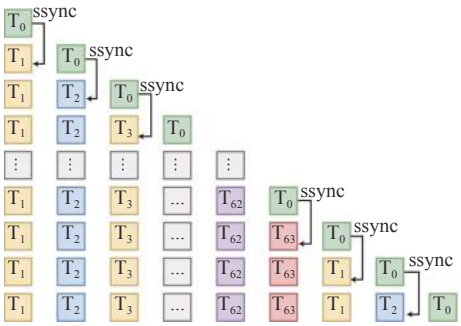


图 7 线程同步示意图

如上文所述, TRSV 包含矩阵-向量乘与求和操作, 它们分别由 1 至 63 号从核和 0 号从核完成 (算法 4 的第 8 行和第 12 行). 基于这样的任务分配方式, 0 号从核需要从其他从核获取矩阵-向量乘的结果. 因此, 为了在从核之间高效地传输数据, 我们设计了基于 RMA 通信机制的归约策略. 图 8 是从核行/列归约示意图. 从上至下依次展示了当前行/列的从核数量为 8, 7, 6, 5, 4, 3, 2 时, 每个从核的数据收发状态. 我们采用 RMA 点对点通信机制先后进行从核行列归约, 最终将每个从核的数据归约至 0 号从核. 算法 5、6、7 是从核归约的伪代码, 其中 l 是归约的数据大小, t 是参与本次归约的线程数量, id 表示从核的线程号, id_{row} 和 id_{col} 分别表示线程的行号和列号. 线程以阻塞方式发起 RMA 写操作 (算法 5 的第 5 行), 由远程回答字判断操作是否完成 (算法 5 的第 9 行), 保证了 RMA 操作有序进行. 由于线程在发起 RMA 操作之前, 已和目标线程同步 (算法 5 的第 4 行), 因此, 线程 T(*, 0) 完成行归约后, 不需相互等待就可进行列归约 (算法 7 的第 10 行).

算法 5. 从核行归约过程 $RowReduction(*y, *y\_0, l, t)$.

1. $\alpha \leftarrow t$ $\beta \leftarrow \lfloor (t+1)/2 \rfloor$
2. **while** $\alpha > 1$ **do**
3. 满足 ($\beta \leqslant id_{\text{col}} < \alpha$) 的从核进行以下操作{
4. 对 $(id-\beta)$ 号从核发起点对点同步
5. 通过 RMA 方式向 $(id-\beta)$ 号从核发送数据
6. }
7. 满足 ($id_{\text{col}} < \beta$ 且 $id_{\text{col}} < \alpha - \beta$) 的从核进行以下操作{
8. 响应 $(id+\beta)$ 号从核的点对点同步请求
9. 判断是否已收到 $(id+\beta)$ 号从核发送的数据, 若是, 进行求和操作; 若不是, 则继续等待
10. }
11. $\alpha \leftarrow \beta$ $\beta \leftarrow \lfloor (\alpha+1)/2 \rfloor$

算法 6. 从核列归约过程 $ColReduction(*y, *y\_0, l, t)$.

1. $\alpha \leftarrow t$ $\beta \leftarrow \lfloor (\mathrm{t}+1)/2 \rfloor$
2. **while** $\alpha > 1$ **do**
3. 满足 ($\beta \leqslant id_{\text{row}} < \alpha$) 的从核进行以下操作{
4. 对 $(id-(\beta \times 8))$ 号从核发起点对点同步
5. 通过 RMA 方式向 $(id-(\beta \times 8))$ 号从核发送数据
6. }
7. 满足 ($id_{\text{row}} < \beta$ 且 $id_{\text{row}} < \alpha - \beta$) 的从核进行以下操作{
8. 响应 $(id+(\beta \times 8))$ 号从核的点对点同步请求
9. 判断是否已收到 $(id-(\beta \times 8))$ 号从核发送的数据, 若是, 进行求和操作; 若不是, 则继续等待
10. }
11. $\alpha \leftarrow \beta$ $\beta \leftarrow \lfloor (\alpha+1)/2 \rfloor$

算法 7. 从核归约过程 $Reduction(*y, *buf\_y\_0, l, t)$.

1. $nrows \leftarrow \lfloor (t+7)/8 \rfloor$ //参与归约的从核行数
2. $ncols \leftarrow t \bmod 8$ //最后一行参与归约的从核数
3. 若当前行参与归约的从核数等于 8, 则{
4. $RowReduction(y, buf\_y\_0, l, 8)$ //行归约函数
5. }

6. 若当前行参与归约的从核数小于 8, 则{
7. *RowReduction*(*y*,*buf_y_0*,*l*,*ncols*) //行归约函数
8. }
9. 列号为 0 的从核进行列归约, 即{
10. *ColReduction*(*y*,*buf_y_0*,*l*,*nrows*) //列归约函数
11. }

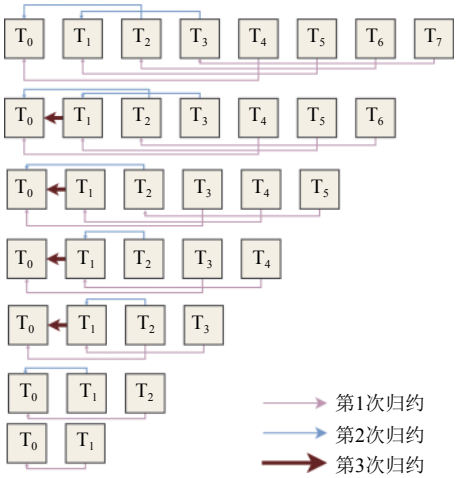


图 8 从核行/列归约示意图

3 访存操作 (DMA) 自适应优化

如上文所述, BLAS 1、2 级函数属于访存密集型问题, 其性能受限于系统访存带宽. 在 SW26010-Pro 处理器上 BLAS 1、2 级函数的访存操作都是通过 DMA 实现的, 函数性能由 DMA 访存带宽决定. 因此, 为了提高函数性能, 有必要针对矩阵/向量规模选择最优的 DMA 参数.

3.1 BLAS 1 级函数

BLAS 1 级函数只处理向量这一种对象, 对向量的 DMA 操作都是连续的, 它的访存带宽取决于单从核进行 DMA 操作时数据量的大小 (见第 1.2 节). 在我们的 BLAS 1 级函数算法中, 每个从核获得大小为 n/t 的连续向量段, 其中 n 是向量规模, t 是参与本次运算的线程数量. 可见, 当数据规模一定时, 从核的 DMA 访存带宽只取决于 t. 如果不对函数进行自适应优化, 当 n 较小, t 较大时, 会造成 DMA 访存带宽偏低. 为了提高 BLAS 1 级函数的性能, 有必要根据向量规模为其设置合适的线程数量.

通过分析 17 个 BLAS 1 级函数的访存和计算特点, 我们发现, BLAS 1 级函数存在两种运行模式, 一是只有访存操作, 包括 COPY 和 SWAP; 另一种有访存和计算两种操作, 包括 AXPY、ROT、SCAL 等.

对于只有访存操作的模式, 由于时间几乎全部花在访存上, 所以我们采用只涉及访存操作的 COPY 作为基准程序, 来预测该模式下 BLAS 1 级函数在问题规模一定的情况下, 最合理的线程设置方案. 为此, 我们设计了如下实验: 在数据规模一定, 线程数量分别为 1、2、4、8、16、32、64 的情况下, 测试 COPY 函数的访存带宽, 实验结果如图 9. 从图中可看出, DMA 的访存带宽随着向量规模的增大而增大并逐渐趋向稳定.

对于有访存和计算两种操作的模式, 由于其他 BLAS 1 级函数如 ROT、SCAL 等与 AXPY 具有相同的执行模式, 即: DMA 读计算 DMA 写. 因此, 我们采用 AXPY 作为另一个基准程序来预测该模式下 BLAS 1 级函数在问题规模一定的情况下, 最合理的线程设置方案. 我们进行了另一个实验: 在数据规模一定, 线程数量分别为 1、2、4、8、16、32、64 的情况下, 测试 AXPY 的访存带宽, 实验结果如图 10. 图 10 中函数 AXPY 的访存带宽与图 9 中函

数 COPY 的访存带宽具有相同的规律, 随着向量规模的增大而增大并逐渐趋向稳定, 当 T=32, 为 18 至 20 时, 访存性能出现明显下降与内存控制器的合并效率下降有关.

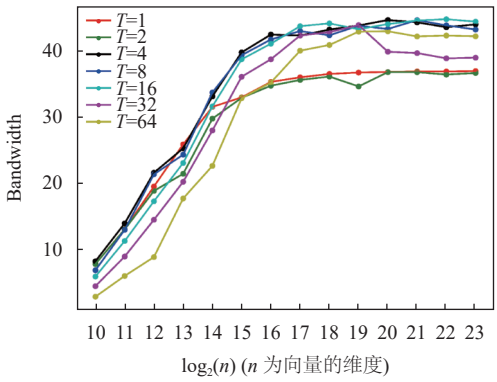


图 9 COPY 的访存带宽

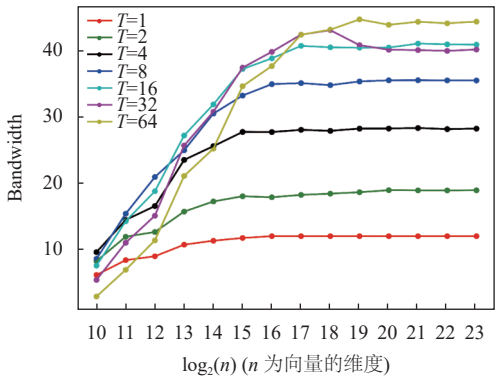


图 10 AXPY 的访存带宽

我们根据图 9、图 10 的结果按最佳访存配置分别为 COPY、SWAP 和 AXPY、ROT 等函数设置了合适的线程数量, 具体的策略如表 1 和表 2. 对于小规模的函数, 考虑到从核启动开销较大 (占总时间的 10%), 从核性能相较于主核性能不占优势, 采用主核进行计算.

表 1 COPY、SWAP 等函数的各个向量规模应设置的线程数

向量规模 (n)	线程数量 (t)
$1 \leqslant n < 1024$	采用主核进行操作
$1024 \leqslant n \leqslant 65536$	4
$n > 65536$	16

表 2 AXPY、ASUM、DOT、MAX、NRM2 等函数的各个向量规模应设置的线程数

向量规模 (n)	线程数量 (t)
$1 \leqslant n < 1024$	采用主核进行计算
$1024 \leqslant n \leqslant 4096$	8
$4096 < n \leqslant 32768$	16
$32768 < n \leqslant 262144$	32
$n > 262144$	64

3.2 BLAS 2 级函数

BLAS 2 级函数的主要操作对象是矩阵, 矩阵的 DMA 操作往往是非连续的. 如第 1.2 节所述, DMA 访存带宽取决于单从核对矩阵进行 DMA 操作时跨步向量块 (*bsize*) 的大小. 由于其他 BLAS 2 级函数如 GER、TRSV 等与 GEMV 具有相同的执行模式即: DMA 读计算 DMA 写. 因此, 我们采用 GEMV 作为基准程序, 来预测 BLAS 2 级函数在问题规模一定的情况下, 进行 DMA 操作的最优. 在我们的 BLAS 2 级函数算法中, 需要矩阵 A、向量 x 和向量 y 这 3 个数组的值才能完成计算, 而每个从核的 LDM 空间仅有 256 KB. 假设 x 和 y 需要的 LDM 空间大小为和, 则 A 需要大小为的空间 (是 A 进行 DMA 操作的), 它们进行 DMA 操作时应满足不等式 (以双精度为例). 因此, 满足条件的和组合有 (8, 2048)、(16, 1024)、(32, 512)、(64, 256)、(128, 128)、(256, 64)、(512, 32)、(1024, 16)、(2048, 8). 我们进行了一个实验, 在矩阵规模一定, 不同组合下, 测试单从核下 GEMV 的访存带宽. 实验结果如图 11所示, 从图中可看出, 当和为 (128, 128) 时, DMA 访存带宽最高. 因为对于 GEMV 来说, 越大, A 的访存带宽越高, 函数性能越好. 当增大到一定程度时, A 的访存带宽基本趋于稳定, 若再增大, 会偏低, 会造成 x 访存带宽下降, 函数性能降低. 多从核下的 GEMV 的实验结果与单从核的基本吻合, 如图 12. 所以我们选取的最优为 128.

BLAS 2 级函数的最优 *bsize* 被确定后, 线程数量是可以根据其问题规模来调整的唯一参数. 为了预测 BLAS 2 级函数在问题规模一定的情况下, 最合理的线程设置方案, 我们设计了一个实验: 在数据规模一定, *bsize* 为 128, 线程数量分别为 1、2、4、8、16、32、64 的情况下, 测试 GEMV 的访存带宽. 实验结果如图 13. 我们根据图 13 的结果为 BLAS 2 级函数的各个规模设置合适的线程数量. 即: 当矩阵规模处于 (128×128, 2048×2048] 区间时, 启

动 16 个线程; 当矩阵规模大于 2048×2048 时, 启动 64 个线程; 当矩阵规模小于 128×128 时, 考虑到从核启动开销较大 (占总时间的 10%), 从核性能相较于主核性能不占优势, 采用主核进行计算.

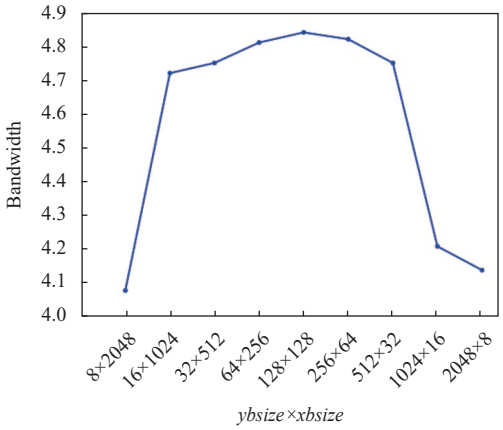


图 11 GEMV 的访存带宽 (t=1)

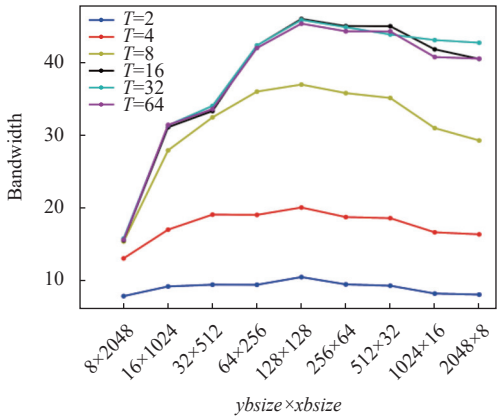


图 12 GEMV 的访存带宽 (t>1)

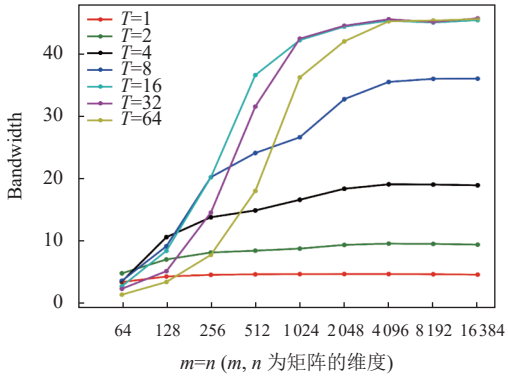


图 13 GEMV 的访存带宽

4 实验结果及分析

4.1 对比实验

针对本文中使用的向量压缩、向量化技术等优化手段, 本文在 SW26010-Pro 平台上进行了一系列对比实验. 对于向量压缩技术, 本文以 GEMV 为实验对象, 对比它是否采用向量压缩操作两种情况下的性能结果, 向量的增量 *incx*、*incy* 分别设置为 *incx*=17, *incy*=19; 对于向量化技术, 本文以 IAMAX 为实验对象, 对比它是否采用向量化计算两种情况下的性能结果; 对于动态任务划分, 本文以 SYMV 为实验对象, 对比它采用静态任务划分和动态任务划分两种情况下的性能结果. 实验中的数据类型为双精度.

后文图 14 是 GEMV 的性能对比结果, 由图可知, 采用向量压缩技术之后, GEMV 性能有明显的提升. 后文图 15 是 IAMAX 的性能对比结果, 由图可知, 采用向量化技术之后, 函数性能得到了 1–2 倍的加速. 后文图 16 是 SYMV 的性能对比结果, 由图可知, 相比静态任务划分方式, 动态任务划分方式使 SYMV 性能有 6%–35% 的提升. 可见本文提出的优化技术给函数性能带来了明显的提升. 在下面的实验中, 本文将展示所有优化技术均应用于 BLAS 1、2 级函数时, 函数性能的提升效果.

4.2 性能结果

BLAS 1、2 级函数的性能受限于系统访存带宽, 本文以从核访存带宽和开源库的加速比作为 BLAS 性能的主

要评估依据. 在实验中, 我们选取的问题规模保证了两个版本的函数性能均达到了各自的最优值, 数据类型为双精度, 执行时间取 1 000 次测量的均值. 对于 BLAS 1 级函数, 访存带宽的计算以向量本身大小作为基准访存量; 对于 BLAS 2 级函数, 访存带宽的计算以矩阵和向量数据量之和作为基准访存量. 该平台实测访存带宽最大约为 46 GB/s.

表 3 是 BLAS 1、2 级函数的访存带宽测试及加速比结果. 表中仅涉及内存读写操作的 COPY 可以实现峰值带宽的 95%; 既涉及内存读写又涉及计算操作的 AXPY 和 SCAL 可以实现峰值带宽的 90%; 涉及普通矩阵的 GEMV 和 GER 可以分别实现峰值带宽的 99% 和 91%; 涉及带状矩阵的 GBMV 可以实现峰值带宽的 87%; 涉及三角矩阵的 TRMV 可以实现峰值带宽的 96%; BLAS 1 级函数的平均访存带宽为 42.07 GB/s, BLAS 2 级函数的平均访存带宽为 37.04 GB/s. 可见, 在本文的优化方案下, BLAS 1、2 级函数可以充分利用从核 DMA 访存带宽.

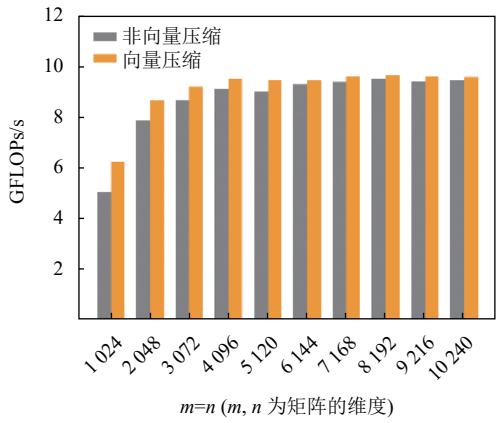


图 14 GEMV 性能对比图

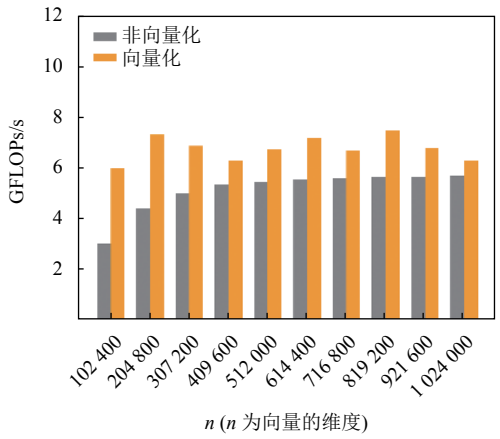


图 15 IAMAX 性能对比图

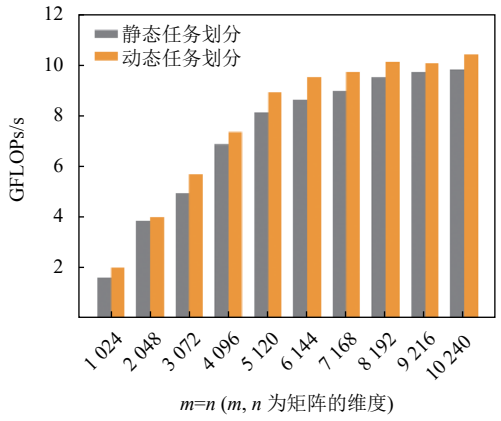


图 16 SYMV 性能对比图

表 3 BLAS 1、2 级函数的访存带宽测试数据及加速比

函数名	带宽 (GB/s)	加速比	函数名	带宽 (GB/s)	加速比
ASUM	41.14	15.35	TRSV	43.13	45.33
DOT	43.04	14.48	GBMV	40.26	25.73
IAMAX	41.91	58.95	SPMV	34.05	8.07
NRM2	41.11	8.83	SPR	31.82	6.05
AXPY	41.76	18.64	SPR2	32.19	6.80
COPY	43.67	14.97	SYMV	35.85	65.47
ROT	40.51	13.44	SYR	31.28	8.63
SCAL	41.79	8.19	SYR2	30.81	11.97
SWAP	43.70	12.77	TRMV	44.42	46.53
GEMV	45.65	51.74	TPSV	33.05	14.60
GER	43.93	12.19	TPMV	35.09	15.42

表 3 中也显示了本文设计的 BLAS 库与开源库 GotoBLAS 0.2.6[22]的性能加速比, 从实验结果可看出, 每个函数对比 GotoBLAS 都有不同程度的加速. 本文实现的 BLAS 1 级函数对比 GotoBLAS 的平均加速比为 18.78, BLAS 2 级函数对比 GotoBLAS 的平均加速比为 25.96. 可见, 本文提出的优化方案能够在申威众核计算平台为 BLAS 1、2 级函数带来明显的性能提升.

4.3 自适应优化

为了验证自适应优化的合理性, 我们以 SCAL 和 TRMV 为实验对象, 测试了它们在进行自适应优化后的性能结果, 并与 t=64 (t 为线程数) 的结果进行了对比. 在实验中, 向量的规模为 1024–1048576, 矩阵的规模为 256–16384,

数据类型为双精度.

图 17 和图 18 分别是 SCAL 和 TRMV 的访存带宽测试结果, 图中的实线是自适应优化的性能结果, 虚线是 t=64 对应的性能结果. 通过观察分析可知, 自适应优化取得了比较明显的加速效果, 平均性能提升为 50%.

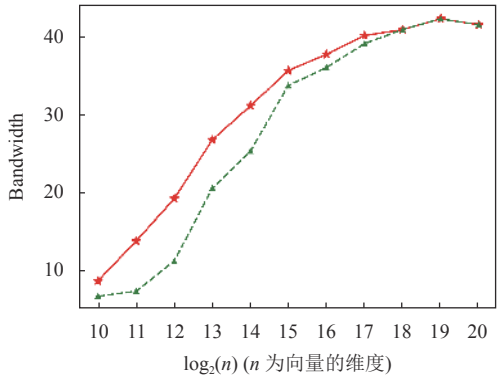


图 17 SCAL 的访存带宽

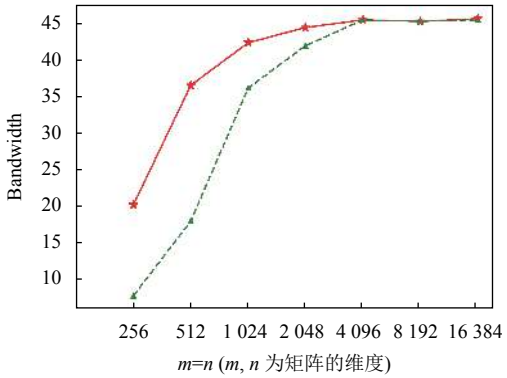


图 18 TRMV 的访存带宽

4.4 相关应用

本文以线性代数中的 LQ 分解 (LQ decomposition)[13]、QR 分解 (QR decomposition)[14]、特征值问题 (eigenvalue problems)[15,16]为实验对象, 验证本文的工作在应用中的优化效果. 我们以 LAPACK 函数 GELQF[39]、GEQRF[39]以及 SYEVD[39]为例, 测试它们调用本文优化的 BLAS 1、2 级函数和调用 GotoBLAS 0.2.6[22]的 BLAS 1、2 级函数的性能数据. 实验中上述 3 个函数调用的 BLAS 3 级函数均来源于 GotoBLAS 0.2.6[22], 数据类型为双精度, 实验结果如表 4.

表 4 LQ 分解、QR 分解及对称特征问题性能对比 (GFLOPs/s)

矩阵规模	调用本文的BLAS			调用GotoBLAS			加速比		
	LQ分解	QR分解	对称特征问题	LQ分解	QR分解	对称特征问题	LQ分解	QR分解	对称特征问题
1024×1024	4.57	14.86	6.57	0.41	1.58	0.91	11.15	9.41	7.22
2048×2048	11.30	31.85	13.37	0.89	3.13	1.89	12.70	10.18	7.07
4096×4096	21.97	64.66	27.6	1.54	5.69	2.54	14.27	11.36	10.87
8192×8192	41.17	134.90	55.17	2.68	10.76	5.68	15.36	12.54	9.71

表 4 是 GELQF、GEQRF 以及 SYEVD 的性能测试结果, 它们通过调用本文优化的 BLAS 1、2 级函数, 取得了平均 10.99 倍的加速效果, 充分说明了本文工作在应用中的重要性.

5 总 结

本文基于 SW26010-Pro 众核计算平台, 提出了若干 BLAS 1、2 级函数的并行算法, 并对不同类型函数针对性地进行了优化, 实验结果说明本文提出的优化方案为 BLAS 1、2 级函数带来了明显的性能提升.

从申威 1600 到申威 26010 再到最新的申威 26010-Pro, 可以看出国产高性能计算平台更新迭代较快, 而 BLAS 1、2 级函数数量较多同时优化方法各异, 同时 BLAS 1、2 级函数属于访存受限型计算, 如何针对该系列平台设计出充分利用平台访存带宽且可以一定程度上利用自动代码生成技术的 BLAS 1、2 级函数, 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] Choi J, Dongarra JJ, Susan Ostrouchoy L, Petitet AP, Walker DW, Clint Whaley R. Design and implementation of the scaLAPACK LU, QR, and cholesky factorization routines. Scientific Programming, 1996, 5(3): 483083. [doi: 10.1155/1996/483083]

[3] King DE. Dlib-ml: A machine learning toolkit. The Journal of Machine Learning Research, 2009, 10(3): 1755–1758.

[4] Demšar J, Curk T, Erjavec A, Gorup Č, Hočevar T, Milutinovič M, Možina M, Polajnar M, Toplak M, Starič A, Štajdohar M, Umek L, Žagar L, Žbontar J, Žitnik M, Zupan B. Orange: Data mining toolbox in Python. Journal of Machine Learning Research, 2013, 14(1): 2349–2353. [doi: 10.5555/2567709.2567736]

[5] Messer OEB, Harris JA, Parete-Koon S, Chertkow MA. Multicore and accelerator development for a leadership-class stellar astrophysics code. In: Proc. of the 11th Int'l Conf. on Applied Parallel and Scientific Computing. Helsinki: Springer, 2012. 92–106. [doi: 10.1007/978-3-642-36803-5_6]

[6] Auer AA, Baumgartner G, Bernholdt DE, Bibireata A, Choppella V, Cociorva D, Gao XY, Harrison R, Krishnamoorthy S, Krishnan S, Lam CC, Lu QD, Nooijen M, Pitzer R, Ramanujam J, Sadayappan P, Sibiryakov A. Automatic code generation for many-body electronic structure methods: The tensor contraction engine. Molecular Physics, 2006, 104(2): 211–228. [doi: 10.1080/00268970500275780]

[7] Ao YL, Yang C, Wang XL, Xue W, Fu HH, Liu FF, Gan L, Xu P, Ma WJ. 26 PFLOPS stencil computations for atmospheric modeling on sunway taihulight. In: Proc. of the 31st IEEE Int'l Parallel and Distributed Processing Symp. Orlando: IEEE, 2017. 535–544. [doi: 10.1109/IPDPS.2017.9]

[8] Cook KH. Large-scale atmospheric dynamics and sahelian precipitation. Journal of Climate, 1997, 10(6): 1137–1152. [doi: 10.1175/1520-0442(1997)010<1137:LSADAS>2.0.CO;2]

[9] Abhyankar S, Betrie G, Maldonado DA, Mcinnes LC, Smith B, Zhang H. PETSc DMNetwork: A library for scalable network PDE-based multiphysics simulations. ACM Trans. on Mathematical Software, 2020, 46(1): 5. [doi: 10.1145/3344587]

[10] Dongarra JJ, Luszczek P, Petite A. The LINPACK benchmark: Past, present and future. Concurrency and Computation: Practice and Experience, 2003, 15(9): 803–820. [doi: 10.1002/cpe.728]

[11] Choi J, Demmel J, Dhillon I, Dongarra J, Ostrouchova S, Petitet A, Stanley K, Walker D, Whaley RC. ScaLAPACK: A portable linear algebra library for distributed memory computers—Design issues and performance. Computer Physics Communications, 1996, 97(1–2): 1–15. [doi: 10.1016/0010-4655(96)00017-3]

[12] Lee DD, Seung HS. Algorithms for non-negative matrix factorization. In: Proc. of the 13th Int'l Conf. on Neural Information Processing Systems. Cambridge: MIT Press, 2001. 535–541. [doi: 10.5555/3008751.3008829]

[13] Guo JJ, Qiu BB, Yang M, Zhang YN. Zhang neural network model for solving LQ decomposition problem of dynamic matrix with application to mobile object localization. In: Proc. of the 2021 Int'l Joint Conf. on Neural Networks. Shenzhen: IEEE, 2021. 1–6. [doi: 10.1109/IJCNN52387.2021.9533326]

[14] Zheng YM, Xu AB. Tensor completion via tensor QR decomposition and $L_{2,1}$-norm minimization. Signal Processing, 2021, 189: 108240. [doi: 10.1016/j.sigpro.2021.108240]

[15] Syrocki Ł, Pestka G. Implementation of algebraic procedures on the GPU using CUDA architecture on the example of generalized eigenvalue problem. Open Computer Science, 2016, 6(1): 79–90. [doi: 10.1515/comp-2016-0006]

[16] Lin CP, Lu D, Bai ZJ. Backward stability of explicit external deflation for the symmetric eigenvalue problem. arXiv:2105.01298v1, 2021.

[17] Sørensen HHB. Auto-tuning of level 1 and level 2 BLAS for GPUS. Concurrency and Computation: Practice and Experience, 2013, 25(8): 1183–1198. [doi: 10.1002/cpe.2916]

[18] He SS, Gu NJ, Zhu HT, Liu YJ. Optimization of BLAS for Loongson-3A Architecture. Journal of Chinese Computer Systems, 2012, 33(3): 571–575 (in Chinese with English abstract). [doi: 10.3969/j.issn.1000-1220.2012.03.024]

[19] Mukunoki D, Imamura T, Takahashi D. Fast implementation of general matrix-vector multiplication (GEMV) on kepler GPUS. In: Proc. of the 23rd Euromicro Int'l Conf. on Parallel, Distributed, and Network-based Processing. Turku: IEEE, 2015. 642–650. [doi: 10.1109/PDP.2015.66]

[20] 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 6th Int'l Conf. on Parallel Processing. Bristol: IEEE, 2017. 422–431. [doi: 10.1109/ICPP.2017.51]

[21] Wang XL, Xu P, Xue W, Ao YL, Yang C, Fu HH, Gan L, Yang GW, Zheng WM. A fast sparse triangular solver for structured-grid problems on Sunway many-core processor SW26010. In: Proc. of the 47th Int'l Conf. on Parallel Proceeding. Eugene: Association for Computing Machinery, 2018. 53. [doi: 10.1145/3225058.3225071]

[22] Smith TM, van de Geijn R, Smelyanskiy M, Hammond JR, van Zee FG. Anatomy of high-performance many-threaded matrix multiplication. In: Proc. of the 28th IEEE Int'l Parallel and Distributed Processing Symp. Phoenix: IEEE, 2014. 1049–1059. [doi: 10.1109

/IPDPS.2014.110]

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

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

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

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

[27] 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]

[28] Wu SG, Xu JF, Yang YZ, Ren G. Implementing of the high performance of BLAS on Beowulf class cluster of computers. Mini-micro Systems, 2001, 22(8): 897–900 (in Chinese with English abstract). [doi: 10.3969/j.issn.1000-1220.2001.08.001]

[29] Gu NJ, Li K, Chen GL, Wu C. Optimization of BLAS based on Loongson 2F architecture. Journal of University of Science and Technology of China, 2008, 38(7): 854–859 (in Chinese with English abstract).

[30] 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]

[31] Sun JD, Sun Q, Deng P, Yang C. Research on the optimization of BLAS level 1 and 2 functions on shenwei many-core processor. Computer Systems & Applications, 2017, 26(11): 101–108 (in Chinese with English abstract). [doi: 10.15888/j.cnki.csa.006045]

[32] Yin J, Yu H, Xu WZ, Wang YX, Tian Z, Zhang YP, Chen BC. Highly parallel GEMV with register blocking method on GPU architecture. Journal of Visual Communication and Image Representation, 2014, 25(7): 1566–1573. [doi: 10.1016/j.jvcir.2014.06.002]

[33] Xu WZ, Liu ZY, Wu J, Ye XC, Jiao S, Wang D, Song FL, Fan DR. Auto-tuning GEMV on many-core GPU. In: Proc. of the 18th IEEE Int'l Conf. on Parallel and Distributed Systems-ICPADS. Singapore: IEEE, 2012. 30–36. [doi: 10.1109/ICPADS.2012.15]

[34] Nath R, Tomov S, Dong TT, Dongarra J. Optimizing symmetric dense matrix-vector multiplication on GPUs. In: Proc. of the 2011 Int'l Conf. for High Performance Computing, Networking, Storage and Analysis. Seattle: ACM, 2011. 6. [doi: 10.1145/2063384.2063392]

[35] Li Y, He SS, Li K. Optimization of BLAS level 2 based on multi-core Loongson 3A. Computer Systems & Applications, 2011, 20(1): 163–167 (in Chinese with English abstract). [doi: 10.3969/j.issn.1003-3254.2011.01.035]

[36] Chohra C, Langlois P, Parello D. Efficiency of reproducible level 1 BLAS. In: Nehmeier M, Von Gudenberg JW, Tucker W, eds. Scientific Computing, Computer Arithmetic, and Validated Numerics. Cham: Springer, 2016, 9553: 99–108. [doi: 10.1007%2F978-3-319-31769-4_8]

[37] Imamura T, Yamada S, Machida M. A high performance SYMV kernel on a fermi-core GPU. In: Daydé MJ, Marques O, Nakajima K, eds. High Performance Computing for Computational Science-VECPAR 2012. Berlin: Springer, 2012, 7851: 59–71. [doi: 10.1007%2F978-3-642-38718-0_9]

[38] Wang Q. Research on dense linear algebra subroutines optimization and automatic code generation on multi-core cpus [Ph.D. Thesis]. Beijing: University of Chinese Academy of Sciences, 2015 (in Chinese with English abstract).

[39] Anderson E, Bai Z, Bischof C, Blackford LS, Demmel J, Dongarra JJ, Du Croz J, Greenbaum A, Hammarling S, McKenney A, Sorensen D. LAPACK Users' Guide. 3rd ed., Philadelphia: Society for Industrial and Applied Mathematics, 1999. 1–404.

附中文参考文献:

[18] 何颂颂, 顾乃杰, 朱海涛, 刘燕君. 面向龙芯3A体系结构的BLAS库优化. 小型微型计算机系统, 2012, 33(3): 571–575. [doi: 10.3969/j.issn.1000-1220.2012.03.024]

[28] 吴少刚, 许解峰, 杨耀忠, 任钢. 高性能BLAS在类Beowulf机群系统上的实现. 小型微型计算机系统, 2001, 22(8): 897–900. [doi: 10.3969/j.issn.1000-1220.2001.08.001]

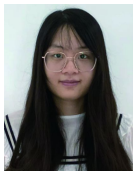
[29] 顾乃杰, 李凯, 陈国良, 吴超. 基于龙芯2F体系结构的BLAS库优化. 中国科学技术大学学报, 2008, 38(7): 854–859.

[30] 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]

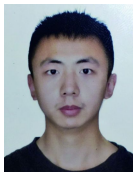
[31] 孙家栋, 孙乔, 邓攀, 杨超. 基于申威众核处理器的1、2级BLAS函数优化研究. 计算机系统应用, 2017, 26(11): 101–108. [doi: 10.15888/j.cnki.csa.006045]

[35] 李毅, 何颂颂, 李恺. 多核龙芯3A上二级BLAS库的优化. 计算机系统应用, 2011, 20(1): 163–167. [doi: 10.3969/j.issn.1003-3254.2011.01.035]

[38] 王茜. 多核CPU上稠密线性代数函数优化及自动代码生成研究 [博士学位论文]. 北京: 中国科学院大学. 2015.



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, 稠密矩阵的相关算法研究.



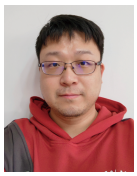
陈道琨(1994－), 男, 博士生, 主要研究领域为高性能计算, 异构并行, 稀疏矩阵的相关算法研究.

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

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.

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.

尹万旺(1980－), 男, 副研究员, 主要研究领域为高性能计算, 数值模拟, 并行调试.



袁欣辉(1989－), 男, 助理研究员, 主要研究领域为软硬件协同设计, 并行算法设计与优化.

林蓉芬(1984－), 女, 工程师, 主要研究领域为高性能计算及其应用.