基于申威 1600 的 3 级 BLAS GEMM 函数优化①

刘　昊[1,2], 刘芳芳[1], 张　鹏[1,2], 杨　超[1], 蒋丽娟[1,2]

[1](中国科学院 软件研究所, 北京 100190)
[2](中国科学院大学, 北京 100190)

摘　要: BLAS 是当前科学计算领域重要的底层支持数学库之一, 其中的 3 级 BLAS 函数应用最为广泛. 本文基于国产申威 1600 平台, 提出了一种基础线性代数库 BLAS 的三级函数通用矩阵乘 GEMM 的高性能实现方法. 在单核上, 使用乘加指令、循环展开、软件流水线指令重排、SIMD 向量化运算、寄存器分块技术等与平台架构相关的技术手段, 实现汇编级手工优化; 在多核上, 提出了适用于该平台的多线程加速方案. 实验结果显示, 在单核串行性能测试中, 与知名开源数学库 GotoBLAS 相比, 我们实现了平均 4.72 倍的加速效果; 在多核并行扩展测试中, 4 线程版的性能则平均达到了单线程版性能的 3.02 倍.

关键词: 申威 1600; 三级 BLAS; GEMM; 高性能计算; 多核

Optimization of BLAS Level 3 Functions on SW1600

LIU Hao[1,2], LIU Fang-Fang[1], ZHANG Peng[1,2], YANG Chao[1], JIANG Li-Juan[1,2]

[1](Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
[2](University of Chinese Academy of Sciences, Beijing 100190, China)

Abstract: BLAS is one of the most important basic underlying math library for scientific computing, in which the level 3 BLAS functions are most widely used. In this paper, we provide a high-performance method to implement Level 3 BLAS functions based on domestic Sunway 1600 platform. To make it clear, we take GEMM as an example. For the implementation on single-core, we apply many tuning techniques related to the specific platform, such as multiply-add instructions, loop unrolling, software pipelining and instruction rearrangement, SIMD operations, and register blocking to push up the performance. For the multi-core implementation, we propose an efficient multi-threaded method. Compared with GotoBLAS, one of the famous open-source BLAS, the experiments show that our serial single-threaded method achieves a speedup of 4.72. What's more, the average speedup of 4-threaded execution towards the single-threaded one can also reach 3.02.

Key words: Sunway 1600; level 3 BLAS; GEMM; HPC; multi-core

1　引言

1.1 背景介绍

BLAS(Basic Linear Algebra Subprograms)是一个线性代数核心子程序的集合,主要包括向量和矩阵的基本操作. 它是最基本和最重要的数学库之一, 广泛应用于科学工程计算. 目前世界上有关矩阵运算的软件几乎都调用 BLAS 数学库; 重要的稠密线性代数算法软件包(如 EISPACK、LINPACK、LAPACK 和 ScaLAPACK 等)的底层都是以 BLAS 为支撑. BLAS 目前已成为线性代数领域的一个标准 API 库.

BLAS 分成三个等级:

1 级: 进行向量-向量操作;

2 级: 进行向量-矩阵操作;

3 级: 进行矩阵-矩阵操作.

本文主要研究 BLAS 3 级中的通用矩阵相乘 GEMM 函数, 如公式(1)所示, 其中 A, B, C 表示矩阵,

① 基金项目:国家自然科学基金(91530103,91530323)
收稿时间:2016-03-16;收到修改稿时间:2016-04-27　[doi: 10.15888/j.cnki.csa.005456]

α，β 为标量因子.

$$C = \alpha AB + \beta C \quad (1)$$

针对特定的体系结构进行 BLAS 库的优化, 特别是GEMM函数的优化工作, 国内外已有相当多的研究成果, 例如 Goto 等人所提出的矩阵乘法实现算法, 通过对矩阵 A 和 B 进行划分, 矩阵乘法细分为一组 panel-panel 的乘法操作(GEBP), 分析矩阵计算三重循环顺序与 cache 缓存之间的关系模型并提出最优算法[1][2]. 基于该模型的开源高性能 BLAS 数学库有 Texas 大学超级计算中心高性能计算组开发的 GotoBLAS[3]. 近几年来, 针对 BLAS 的自动调优技术日渐成熟, ATLAS 数学库可以针对通用 CPU 平台进行自动调优, 通过自动调整矩阵分块、监测 cache 利用率、调整寄存器使用策略等方式, 生成最合适平台架构的代码[4].

近年来随着 CPU+GPU 异构并行计算的兴起, 基础线性代数库在这些异构平台上的高性能实现也出现大量优秀科研成果[5,6]. 与本文所述的基于稠密矩阵的 BLAS 优化工作不同, 很多基于 STENCIL 计算而出现的稀疏矩阵若以稠密方式进行计算则明显效率低下, 面向这些稀疏矩阵实现高性能的 BLAS 库是近年来该领域的另一个重要方向[7].

神威蓝光国产超级计算机是由国家并行计算机工程研究中心开发研制的首台全部采用国产 CPU 构建的千万亿次超级计算机系统, 处理器采用自主设计生产的申威 1600 CPU, 于 2011 年 10 月份在国家超级计算济南中心投入使用, 其布局按照 MPP 万万亿次架构设计, 主频 975MHz-1.1GHz, 单 CPU 16 核心[8], CPU 总数为 8704, CPU 核心总数接近 14 万, 峰值计算速度达到每秒 1100 万亿次浮点计算, 持续计算能力为 738 万亿次.

随着高性能计算需求的日益增大, 越来越多的应用开始部署在以申威处理器为代表的国产高性能计算平台上. 这些应用中, 无论是在传统的科学计算、气象预报、金融统计等领域, 亦或是现今新兴的机器学习等, 均严重依赖底层 BLAS 库的实现以获得较高的计算性能. 而现有的开源 BLAS 库, 均无法有效的利用国产平台的硬件特性, 实际性能较低, 迫切需要针对国产平台进行优化实现的 BLAS 库出现. 因此, 结合神威蓝光等当前国产多核处理器体系结构特征, 研究 BLAS 数学库, 特别是 3 级 BLAS 函数的高性能实现方法具有重要意义.

1.2 本文组织结构

本文主要针对国家超级计算济南中心的申威 1600 平台进行 BLAS 数学库三级函数的优化研究工作, 以通用矩阵相乘函数 GEMM 为例进行表述. 其中第一章是引言, 介绍平台信息与 BLAS 库优化现状以及本文工作的意义; 第二章以 GEMM 为例, 介绍三级 BLAS 函数特点; 第三章介绍基于单核的 GEMM 优化设计; 第四章介绍基于多核的 GEMM 优化设计; 第五章为实验, 与开源的 GotoBLAS 数学库进行性能比较; 最后是总结与展望.

2 三级BLAS GEMM函数概述

一般来说, 1 级 BLAS 函数计算复杂度为 O(N), 2 级 BLAS 函数计算复杂度为 $O(N^2)$, 3 级 BLAS 函数计算复杂度为 $O(N^3)$, 评价一套 BLAS 数学库性能高低, 3 级 BLAS 的性能是最重要的指标, 而通用矩阵乘法函数 GEMM 作为 3 级 BLAS 最常用的函数, 其性能指标往往也最被关注.

GEMM 实现公式 1 所述的矩阵相乘计算, 其本质是 K-N-M 的三重循环计算, 计算复杂度为 $O(N^3)$. 考虑其访存特性, GEMM 函数读取 A、B、C 矩阵各一次, 写回 C 矩阵一次, 复杂度可表述为 O(2MN+MK+KN), 整体为 $O(N^2)$复杂度, 在矩阵的 K、N、M 三维足够大的情况下, 计算复杂度远大于访存复杂度, 将访存开销隐藏在计算过程中是完全可行的, 理论上 GEMM 函数性能应可以接近平台性能峰值.

目前的通用编译器往往只能进行普适性的优化, 例如循环展开, 利用硬件流水线进行加速, 而不能针对 GEMM 函数特性, 分析数据依赖情况, 实现最优化的计算与访存延迟互相掩盖. 例如著名的 NETLIB BLAS, 一种通用的, 使用 FORTRAN 语言实现的 BLAS 库(一般作为正确性与性能的基准测试程序, http://www.netlib.org/blas/). 该 BLAS 库中的 GEMM 使用最朴素的三重循环实现, 可以认为除了编译器优化外不存在其它任何形式优化. 该版本 BLAS 在本文所述的申威 1600 平台上性能为 280 MFLOPS, 仅仅相当于平台性能峰值的 3.5%, 远不能满足应用对 GEMM 的性能要求[9], 故需要设计高效的分块算法, 结合核心段的需要手工汇编来实现.

3 基于单核的串行优化设计

针对申威 1600 平台单核, 基于其平台的扩展 ALPHA 指令集, 本节提出了 GEMM 函数的高性能优化方案. 该方案基于本文第 2 章所述的 GEMM 函数的

计算密集型特征, 充分考虑申威 1600 平台特性, 最大程度发挥该平台计算能力, 实现该平台上的高性能 GEMM 函数, 合理利用平台的寄存器等硬件资源, 实现计算与访存延迟互相掩盖.

3.1 乘加指令

GEMM 函数实现的计算如公式 1 所示, 对 C 矩阵元素进行乘加操作. GotoBLAS 针对 ALPHA 架构的核心函数的汇编实现中没有实现乘加操作, 乘法与加法分别进行. 在寄存器分块为 4*4 的情况下, 读取 A、B 矩阵各 4 份数据, 计算 16 份 C 矩阵的部分解, 计算访存比达到 4 比 1, 在内层循环充分循环展开后, 平均每 4 条计算指令中插入一条通信指令. 高达 4 比 1 的计算通信会增加寄存器的消耗, 不利于软件流水线的排布.

申威 1600 平台提供 SIMD 乘加指令, 每次计算指令可以实现加法和乘法在一个指令周期完成. 在同样 4*4 的寄存器分块情况下, 计算通信比降低为 2 比 1[10], 计算指令总数减少一半, 理论上可达到 2 倍的性能加速, 内层循环中的中间变量数减少, 对寄存器的消耗降低, 利于软件流水线的排布.

3.2 256 位的 SIMD 向量运算

一般而言, 在不考虑编译优化的情况下, 我们可以使用 intrinsic 函数进行手工向量化, 基于 C 代码就可以取得不错的性能提升. 但是对于性能要求较高的任务, 比如本文讨论的 GEMM 函数, intrinsic 函数并不能胜任, 因为 intrinsic 函数不能控制指令的调度和寄存器分配, 所以我们需要在核心函数中编写向量化的汇编代码[9].

申威 1600 平台提供了 256 位的 SIMD 向量扩展支持. 以实数双精度为例, 每个元素为 8 字节、64 位, 进行向量化扩展后, 配合长度为 256 位的浮点向量化寄存器, 每次顺序读取 4 个 A 矩阵元素, 同时对 4 个 C 矩阵元素进行运算, 理论上性能可实现 4 倍加速.

3.3 循环展开与软件流水线

循环展开是编译器经常使用的优化策略之一, 循环展开后, 最内层循环的单次循环内指令数增加, 从硬件角度看, 更多的指令可以用于乱序发射与乱序执行, 利于硬件流水线发挥作用, CPU 保留站等硬件资源也可更充分得以利用; 从软件角度看, 更多的指令可以用于指令重排, 有助于避免“访存-计算”的数据依赖关系. 一般而言, 访问主存往往有较高的延迟, 要求多次的循环展开, 将访存操作的延迟隐藏于运算操作中.

在实际的核心函数实现中, 我们仍然采用了经典的 N-M-K 的计算顺序, 内层循环的部分代码如下, 其中 VMAX 为乘加指令:

```
VMAD    a0,b1,t04,t04
LDDE    nb1,5*SIZE(B)

VMAD    a4,b0,t01,t01
VLD     na12,12*SIZE(A)

VMAD    a4,b1,t05,t05
VLD     na8,8*SIZE(A)

VMAD    a0,b2,t08,t08
LDDE    nb2,6*SIZE(B)

VMAD    a0,b3,t12,t12
LDDE    nb3,7*SIZE(B)

VMAD    a8,b0,t02,t02
VMAD    a8,b1,t06,t06

VMAD    a4,b2,t09,t09
addl    B,8*SIZE,B
VLD     na0,0*SIZE(A)
```

图 1 核心汇编代码举例

在核心函数设计过程中, 以下几点比较重要: (1)硬件提供两条流水线, 无数据依赖的计算与访存可同时发射, 则指令排布应尽量保证一条计算指令和一条取数指令共同出现, 使得两条流水线都占满; (2)最内层循环需展开足够的次数, 掩盖掉所有的本地访存指令, 考虑实际申威 1600 平台数据从 cache 到寄存器的延迟, 最内层的 K 层循环展开两次即可.

3.4 寄存器分块

寄存器分块的目的是合理安排核心函数中矩阵分块的大小, 在不超过硬件限制的情况下尽可能充分利用寄存器等硬件资源, 实现性能优化.

寄存器的数量决定了寄存器分块的大小. 申威 1600 是扩展的 ALPHA 架构平台, 每个 CPU 拥有 F0 到 F31, 共 32 个逻辑浮点寄存器, 本文选择 4*4 的寄存器分块. 在此分块下, 每轮使用 4 个浮点向量寄存器, 取 4 个 256 位向量长度的 A 矩阵元素, 共 16 个 A 矩阵元素; 使用 4 个浮点向量寄存器, 每个寄存器取 1 个 B 阵元素并扩展成 4 份, 则 4 个寄存器处理 4 个 B 矩阵元素, 计算 16 个 256 位向量长度的 C 矩阵元素, 即 64 个 C 矩阵双精元素的部分解, 由此共使用了 24 个浮点向量寄存器. 此外, F31 寄存器固定存储 0 值, 剩余 7 个逻辑寄存器存储中间变量或用作循环变量, 寄存器资源使用合理且高效.

3.5 数据预取与数据重排

申威 1600 提供特定的预取指令 fetchd, 支持将内

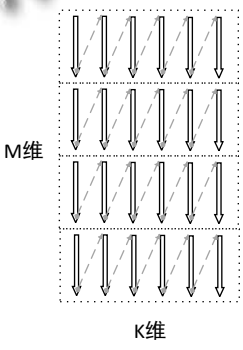
存中的数据预取至 cache 中, 每次预取一个 cache 行的数据量. 若不使用预取指令, 每当一个 cache 行的数据使用完毕, 硬件都需要等待数据从内存加载至 cache 中, 造成流水线停顿, 且这种停顿以百拍来衡量, 对于追求高性能的 GEMM 来说是无法容忍的. 在实测中, 若矩阵 A 拷贝过程不使用预取指令, 总体性能下降 14.1%, 若矩阵B拷贝过程不用预取指令, 总体性能下降43.6%.

使用 fetchd 预取指令的位置及预取跨步的大小需谨慎设计, 若跨步太小, 会造成数据还未完成预取就需要使用, 造成流水线停顿; 若跨步太大, 会造成预取数据完成后长时间没有使用, 再次被替换出去. 申威 1600 CPU 使用的 cache 替换顺序为 FIFO, 即先进先出的策略, 这对 GEMM 函数的设计而言是有利的, cache 数据的替换是可控的, 因此只要测试出合适的跨步, 性能可以保证稳定.

数据的预取仅考虑到指令位置与跨步大小是不够的. 按照 Goto 等人的分析[1,11], GEMM 核心函数三重循环的最优顺序应是 N-M-K, 且最内层循环, 即 K 层循环应保证足够高的性能, 尽量减少延迟, 最大程度利用好 cache 和寄存器等硬件资源, 否则函数整体性能无从谈起[12].

我们以列主元矩阵为例进行说明, 若矩阵数据不进行任何处理直接按照 N-M-K 的顺序进行计算, 使用前文所述的 4*4 寄存器分块, 不考虑 SIMD 向量化影响, 且假设分块矩阵的M维足够大, 则每次cache行只有前 4 个元素能够使用, 沿 K 维进行的下一次计算需要的数据一定不在该 cache 行中, cache 资源被极大浪费, 且极易因 cache 不命中造成流水线停顿.

为此, 需考虑将 A 矩阵和 B 矩阵沿 K 维方向进行重排, 每次 cache 行所取数据都被完全利用, 重排效果如图 2 所示.



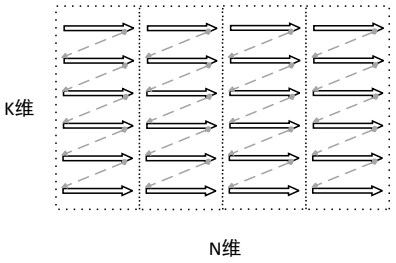


图 2 A 矩阵和 B 矩阵重排示意图

4 基于多核的优化设计

考虑计算任务的划分, 各线程对矩阵进行等份划分, 计算各自对应的C矩阵部分解即可, 理论上讲, 线程之间没有数据依赖关系, 可以说GEMM函数具有天然的多线程负载均衡特性. 对于 GEMM 函数而言, 进行多核并行化优化实施并不复杂, 每个计算核心运行一个线程, 每个线程之上的优化设计仍采用上一章所述的基于单核的优化设计方案即可.

直观来看, 任务划分可以按照A矩阵的M维进行划分或按照 B 矩阵的 N 维进行划分, 本文使用了行方向, 即沿着A矩阵的M维进行划分的方法, 4线程并行化的数据划分, 如图 3 所示, 每个线程计算对应的 C 矩阵部分解.

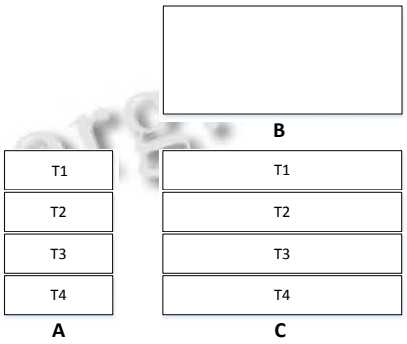


图 3 线程间数据划分

如图3所示, 4个线程各自拥有私有的部分A矩阵, 线程间共享整个B矩阵, 计算对应的C矩阵的部分解. 由于申威 1600 架构支持多线程共享内存, 理论上 4 线程对一个内存地址具有相同的访存延迟, 不会发生某个线程需要等待其他线程访存的情况.

正如前文所述, GEMM 具有天然的负载均衡特性, 在矩阵规模合理, 不考虑特殊规模或边界的情况下, 可以做到所有线程计算规模完全或近似相等. 然而, 由于每个线程内部的计算仍然按照单核进行, 仍需采

用寄存器分块、软件流水线等技术, 每个线程需各自进行 A、B 矩阵的数据重排, 我们对 A、B 矩阵的重排情况分别进行讨论.

首先考察 A 矩阵, 由于每个线程私有部分 A 矩阵元素, 线程内部进行的矩阵重排不会影响到其他线程, 多个线程可以同时进行各自 A 矩阵的数据重排.

接下来考察 B 矩阵, 我们已知所有线程共享 B 矩阵的所有元素, 一种简单有效的方案是所有线程等待 1 号线程完成 B 矩阵的数据重排, 然后再并发进行计算. 但是这种方案破坏了 GEMM 本身具有的负载均衡特性, 在 1 号线程进行数据重排的过程中, 其他线程将进行等待; 另一方面讲, B 矩阵的划分工作从直观上讲是完全可以并发执行的. 因此, 我们实际也需要将 B 矩阵划分于各个线程之中, 如图 4 所示, 将 B 矩阵按列划分于每个线程之中.

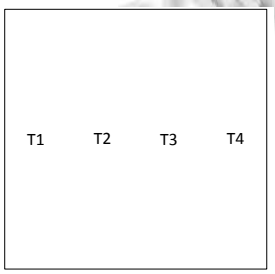


图 4　B 矩阵划分

考虑到 B 矩阵需要共享, 我们设计了一种二维数组的线程间共享的数据结构 working[i], 表示第 j 个线程为第 i 个线程准备的重排后的 B 矩阵元素, working[i]中的每个元素为一个指针, 指向对应的重排后的 B 矩阵元素头指针. 换言之, 重排后的 B 矩阵数据被各个线程私有, 但这些数据指针却是共享的, 基于申威 1600 的平台多线程共享内存架构, 各线程可以相同的延迟共享这些数组.

该共享的数据结构内的元素为指针, 可用于线程间同步而无需另外设计线程间的锁机制, 当 working[i]为空时, 表示此时的对应 B 矩阵还未重排准备就绪, 不能用于计算, 该线程应该等待; 反之, 表示该 B 矩阵已重排准备完毕, 第 i 个线程可以获取第 j 个线程的 B 元素进行计算, 计算完毕后 working[i]置为空, 等待下一轮的第 j 个 B 矩阵重排完毕. 同时, 只有 working[all]都为空时, 第 j 个线程的 B 矩阵才算使用完毕, 没有其他线程需要这部分数据可以释放并进行下一轮的 B 矩阵数据重排.

5　性能测试与评价

申威 1600 平台作为国产高性能多核平台, 性能指标优异, 已被国家超级计算济南中心等国内超算中心使用, 该平台即为本文实验的测试平台. 该平台使用了扩展的 ALPHA 架构指令集, 每个 CPU 支持乘加指令与 256 位向量化运算, 支持计算指令与访存指令同步发射, 具体硬件参数如表 1 所示.

表 1　申威 1600 主要参数

类型	参数
处理器 CPU	SW1600 64bit 16 核心, 主频 1.0GHz
内存容量	170TB
1 级数据 cache	2 路-64KB, 64 Byte line
1 级指令 cache	2 路-64KB, 64 Byte line
编程语言及环境	C、C++、Fortran、JAVA、MPI、OpenMP

表 2 统计了统计基于单核的单线程方案, 数据规模分别为 1024、2048、4096、8192, 申威 1600 GEMM 函数性能和 GotoBLAS GEMM 性能, 如图 5 所示. 实验涵盖了如上四种规模的实数单精、实数双精、复数单精、复数双精共计 16 组测试用例, 平均加速比为 4.72, 最高加速比为 5.61. 为说明基于申威多核的 4 线程设计方案加速效果, 表 3 统计了基于多核的 4 线程方案, 包含了数据规模与数据类型的同上的 16 组测试用例, 与申威平台单核的单线程方案相较, 平均加速比为 3.02, 最高加速比为 3.18. 对比图如图 6 所示.

表 2　申威单核与 GotoBLAS 单核性能对比

数据类型	编号	数据规模	加速比
实数单精	1	1024	5.34
	2	2048	4.95
	3	4096	5.02
	4	8192	4.97
实数双精	5	1024	4.03
	6	2048	4.16
	7	4096	4.10
	8	8192	4.00
复数单精	9	1024	5.61
	10	2048	5.49
	11	4096	5.53
	12	8192	5.16
复数双精	13	1024	4.39
	14	2048	4.47
	15	4096	4.43
	16	8192	4.34

表 3 申威多核 4 线程与申威单核性能对比

数据类型	编号	数据规模	加速比
实数单精	1	1024	2.90
	2	2048	2.99
	3	4096	2.97
	4	8192	2.86
实数双精	5	1024	3.15
	6	2048	3.11
	7	4096	3.13
	8	8192	3.18
复数单精	9	1024	2.59
	10	2048	2.96
	11	4096	2.96
	12	8192	3.17
复数双精	13	1024	2.91
	14	2048	3.16
	15	4096	3.17
	16	8192	3.15

6 总结与展望

本文首先说明了三级 BLAS GEMM 函数的计算密集型的特征, 接下来介绍了国产申威 1600 多核平台的特征. 基于该平台扩展的 ALPHA 架构, 提出了基于该平台的高性能 BLAS 三级函数实现方案. 本方案首先分析了基于单核的单线程优化方案, 提出了矩阵数据重排、软件流水线、寄存器分块等基于该平台的优化技术; 接着分析了基于多核的多线程优化方案, 提出了矩阵划分的方式与多线程同步机制. 在性能测试中, 基于申威 1600 平台优化过的 GEMM 函数性能, 在单核情况下对比 GotoBLAS 的平均加速比为 4.72, 多核 4 线程方案的平均加速比为 3.02, 达到了预期的效果.

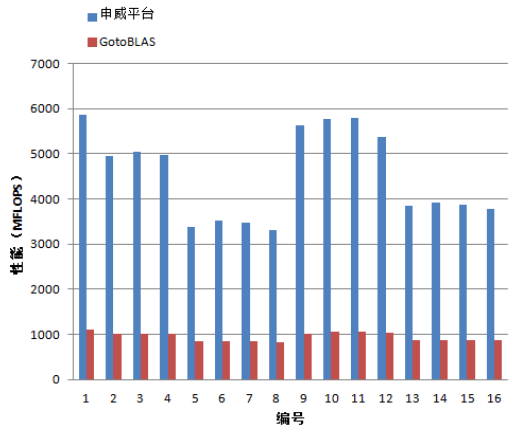


图 5 申威单核与 GotoBLAS 对比

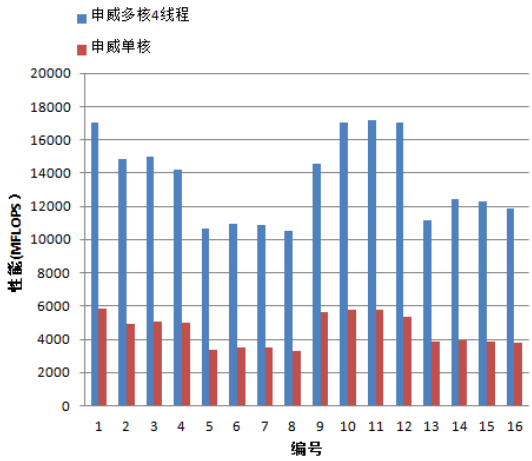


图 6 申威 4 线程与单核对比

申威 1600 作为新兴的国产高性能计算平台, 我们期待着该平台系列的后续机型, 针对该系列平台未来将可能出现的新特性, 这对这些特性又会对高性能的 BLAS 数学库提出怎样的要求, 我们拭目以待.

参考文献

1 Goto K, Geijn RA. Anatomy of high-performance matrix multiplication. ACM Trans. on Mathematical Software (TOMS), 2008, 34(3): 12.

2 Goto K, Van De Geijn R. High-performance implementation of the level-3 BLAS. ACM Trans. on Mathematical Software (TOMS), 2008, 35(1): 4.

3 蒋孟奇,张云泉,宋刚,等.GOTOBLAS 一般矩阵乘法高效实现机制的研究.计算机工程,2008,34(7):84–86,103.

4 Whaley RC, Petitet A, Dongarra JJ. Automated empirical optimizations of software and the ATLAS project. Parallel Computing, 2001, 27(1-2): 3–35.

5 张帅,李涛,王艺峰,等.细粒度任务并行 GPU 通用矩阵乘.计算机工程与科学,2015,37(5):847–856.

6 Kurzak J. Preliminary results of autotuning GEMM kernels for the NVIDIA Kepler architecture-GeForce GTX 680 [z3.I, APACK Working Note 267,2014

7 李佳佳,张秀霞,谭光明,等.选择稀疏矩阵乘法最优存储格式的研究.计算机研究与发展,2014,51(4):882–894.

8 申威 1600 处理器的细枝末节.黑龙江科技信息,2011,(34):I0004–I0004.

9 张先轶.若干线性解法器多核和众核性能优化关键技术研究[学位论文].北京:中国科学院大学,2014.

10 李毅,何颂颂,李恺等.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.

11 张先轶,王茜,张云泉,等.OpenBLAS:龙芯 3A CPU 的高性能 BLAS 库.2011 年全国高性能计算学术年会(HPC china2011)论文集,2011.

12 陈少虎.BLAS 库在龙芯 3A 上的实现与优化[学位论文].北京:中国科学院研究生院,2011.