doi:10.3772/j.issn.1002-0470.2016.10 – 11.001

基于龙芯 3A2000 处理器的高性能 Goto BLAS 库的实现①

张华亮②[*,**,***]　黄启印[*,**]　吴少校[*,***,***]

([*]中国科学院计算技术研究所计算机体系结构国家重点实验室　北京 100190)

([**]中国科学院大学　北京 100049)

([***]龙芯中科技术有限公司　北京 100190)

摘　要　用 Linpack 测试集测试了计算机系统浮点性能,测试用函数运算库为 Goto BLAS 库。该库对 Linpach 的测试结果有很大影响。为了提高 Goto BLAS 性能,观察了 Goto BLAS 库在龙芯 3A2000 处理器平台的性能表现,分析了测试软件的执行流程、数据的处理方法,根据处理器的结构特点,合理配置矩阵分块参数,优化核心循环的实现方案,同时采用软硬件数据预取技术及优化的内核 TLB 配置策略。在这些优化方法的共同作用下,仿真平台上核心函数的浮点部件效率超过 90%。优化方案在本实验中取得了显著的效果。

关键词　Goto BLAS, 性能优化, Linpack, 矩阵运算, 数据预取

0　引　言

线性系统软件包(linear system package, Linpack[1])是用来测试高性能计算机系统的浮点计算性能的工具软件,也是国际上 TOP500 超级计算机的性能评测工具。该测试采用高斯消元法解稠密一元 N 次线性方程组。运算中通过分块递归 LU 分解法处理系数矩阵,将原始的线性方程组转化为上三角方程,通过回代逐一得到方程组的解。该方法的计算部分主要通过调用基本线性函数库(Basic Linear Algebra Subprograms, BLAS)中的函数完成。BLAS 库主要有 Goto BLAS、ATLAS 和 Generic BLAS 三种。本文采用 Goto BLAS 作为 BLAS 库的实现方法。该库考虑到现代处理器的 Cache 和旁路转换缓冲(translation lookaside buffer, TLB)结构在计算中采用分块计算和分块存储的思想,同时针对不同的处理器进行优化,这些特点使得 Goto BLAS 库在测试中得到了广泛的应用。目前基于云计算、分布式集群系统的应用日趋成熟。高性能、大规模的科学计算需求日益增长。Wang 等[2]指出了提高分布式 Linpack 测试性能的 3 个途径:选取并优化 BLAS 库,调整网络性能降低消息传送的延迟,根据物理内存配置选择合适的问题规模。针对具体系统结构的 Goto BLAS 库优化技术对提高整个系统的性能具有重要的意义。

本文以高性能的通用国产处理器龙芯 3A2000[3]为研究平台。通过分析该处理器的结构组成,观察 Linpack 测试在流水线中的具体行为,采用合适的矩阵分块方法,提出了算法核心循环的实现方法,并根据处理器特性提出了数据预取方案及系统相关的 TLB 优化方法。通过这些方法,Linpack 性能得到显著的提升。本实验基于龙芯 3A2000 多核处理器的验证平台进行,但是该优化方法不局限于 MIPS[4]架构的处理器,也适用于其他体系结构的处理器平台。

① "核高基"科技重大专项课题(2014ZX01020201)和 863 计划(2012AA012202, 2013AA014301)资助项目。

② 男,1982 年生,博士;研究方向:高性能计算机体系结构;联系人,E-mail: zhanghualiang@ict.ac.cn

(收稿日期:2016-08-17)

1 相关工作

Linpack 基准测试程序自公布以来,受到工业界及学术界的广泛关注。针对该测试的分析优化方法也成为研究热点。Belter[5]研究了访存带宽对 BLAS 库中核心函数的影响。为了提升函数的性能,他开发了一套专用编译器,通过一个算法选择优化选项并编译 BLAS 库,运行后找到最佳的编译选项组合。他也提出了一种快速评估优化策略的方法。

Wang[6]等人提出了基于异构集群系统的 Linpack 性能优化算法。该系统配置了 GPGPU 组件。该方法重新实现了 HPL 的组件包,并命名为 GHPL。该 GHPL 程序具有良好的扩展性。在 16 个节点组成的测试集群中,每个节点分别采用 Tesla 系列 GPU 和 GeForce GTX 系列的 GPU,该方法使得测试性能提升 2 倍。

为了充分发挥多核系统的性能, Wang[7]采用基于软件中间层的混合 MPI/OpenMP 编程模型解决科学计算中的瓶颈问题,重点研究了局部交换算法并采用多线程技术提升性能,将局部交换时间从 277s 降低到 202s,进而将总体程序的性能提升 5.6%。

李毅[8]等人针对龙芯 3A 的系统结构特点,采用地址交错技术、共享数据的任务划分方法对 BLAS 的多级库进行性能优化调整。同时采用循环展开等软件优化技术提升 Linpack 性能。在实际测试中,单核性能达到 75%,四核处理器效率可以达到 55%。

2 程序性能瓶颈分析

Linpack 的 HPL 程序在龙芯 3A2000 处理器平台的测试中,耗时部分有 LU 分级算法的寻找主元、矩阵的行交换步骤以及 Goto BLAS 库中的计算函数。这些函数包括矩阵乘法功能的 DGEMM 函数、特殊三角矩阵的 DTRSM 乘法函数以及矩阵的行变换功能。其中 DGEMM 函数执行时间超过测试总时间的 80%,为测试程序的核心函数。

2.1 矩阵乘法性能分析

Goto BLAS 库处理大规模矩阵乘法的优化算法兼顾 TLB 和 Cache 的访问性能,提高矩阵计算的数据局部性,降低 Cache 失效发生的次数。对 $\boldsymbol{C}=\boldsymbol{A}\times\boldsymbol{B}$ 的矩阵乘法,DGEMM 函数采用 3 层循环完成。如图 1 所示,中层循环实现完整的矩阵 $\boldsymbol{A}$ 与矩阵 $\boldsymbol{B}$ 的列分块子矩阵 $\boldsymbol{B}_j$ 的乘法运算。外层循环每次更新 $\boldsymbol{B}_j$,直到运算结束。中层循环轮流访问矩阵 $\boldsymbol{A}$ 的子块 $\boldsymbol{A}_i$,每次迭代开始时将矩阵子块 $\boldsymbol{A}_i$ 中的数据复制到连续的内存的区域 $\boldsymbol{Sa}$,然后将矩阵 $\boldsymbol{B}_j$ 的子块 $\boldsymbol{B}_{kj}$的数据复制到内存缓冲器 $\boldsymbol{Sb}$ 的第 k 个位置。然后进入内存循环,反复调用函数 dgemm _ kernel 对缓冲区的矩阵 $\boldsymbol{Sa}$ 和 $\boldsymbol{Sb}_k$ 执行乘法运算。内层循环结束后,得到矩阵 $\boldsymbol{A}_i$ 和 $\boldsymbol{B}_j$ 的乘积,更新结果子矩阵 $\boldsymbol{C}_{ij}$。此时根据处理器 Cache 的 LRU 算法,$\boldsymbol{Sa}$ 和 $\boldsymbol{Sb}$ 内容存在 Cache 中,并且优先级最高。此后进入下一个中层循环,子矩阵 $\boldsymbol{A}_{i+1}$被复制到 $\boldsymbol{Sa}$ 区域中。如果 Cache 产生冲突缺失事件,$\boldsymbol{A}_{i+1}$的数据将替换掉除 $\boldsymbol{Sa}$,$\boldsymbol{Sb}$ 以及子矩阵 $\boldsymbol{C}_{ij}$之外的内容。之后再次调用底层函数 degmm _ kernel 进行乘法运算。

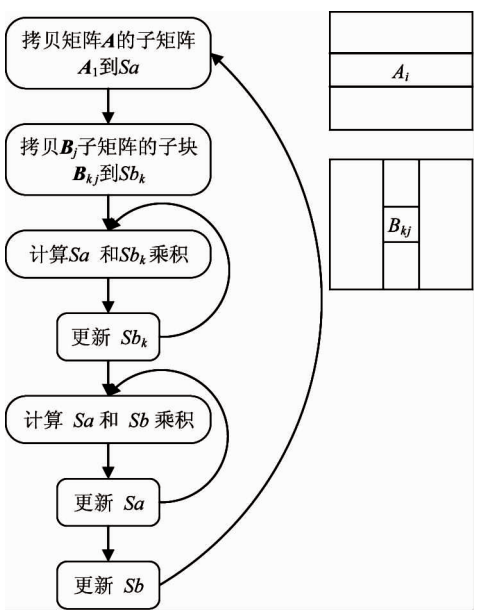

图 1 DGEMM 函数程序的流程图

根据上面的程序流程,在计算功能部件和访存部件为全流水工作的处理器中,当浮点部件满负荷运行时,处理器可达到性能峰值。但是在实验环境

中矩阵乘法操作没有达到理论峰值。主要原因有:

(1) 测试程序采用多层循环方式调用核心计算函数。循环中的分支预测事件可造成流水线等待,使得浮点队列部件无指令发射。此外,循环开始以及结束部分进行相关的数据操作时,浮点功能部件处于空闲状态。

(2)算法操作的指令序列存在数据相关,计算指令等待访存指令的写回结果。在等待期间,浮点功能部件无指令输入,浮点流水线中产生空泡。访存延迟的原因有 Cache 失效和 Cache 替换等。

(3)由于寄存器堆资源耗尽或者等待访存延迟导致浮点发射队列 FTQ 暂时无法发射指令,同时前端流水将译码后的指令调度到该队列,使得发射队列 FTQ 队满。进而导致前端流水线停顿,影响性能。

2.2 矩阵分块性能分析

在 DGEMM 函数的矩阵乘法运算中,分块策略要兼顾数据访问的性能以及每次对分块进行核心计算的效率。

在矩阵乘法中,假设矩阵 $\boldsymbol{A}$ 为 $M\times K$ 的矩阵,矩阵 $\boldsymbol{B}$ 为 $K\times N$ 的矩阵,乘积矩阵 $\boldsymbol{C}$ 为 $M\times N$ 的矩阵。分块算法将矩阵 $\boldsymbol{A}$ 的子块设置为 $P\times Q$ 大小的矩阵,矩阵 $\boldsymbol{B}$ 的子块大小设置为 $Q\times R$。内层循环中计算公式 $\boldsymbol{C}_{ij}+=\boldsymbol{A}_i\times\boldsymbol{B}_j$ 中子矩阵 $\boldsymbol{C}_{ij}$大小为 $P\times R$。根据矩阵乘法公式,总计算次数为 $M\times K\times N$。期间矩阵 $\boldsymbol{A}$ 的搬运次数为 N/R,矩阵 $\boldsymbol{B}$ 只被访问一次。缓存区 $\boldsymbol{Sa}$ 写入次数为$(M\times K)/(P\times Q)\times(N/R)$。缓冲区 $\boldsymbol{Sb}$ 写入次数为$(K/Q)\times(N/R)$次。$\boldsymbol{C}_{ij}$所在缓冲区写入次数为$(K/Q)\times(M/P)\times(N/R)$。完成计算所需的数据搬运量为上面所有参与运算单元的总和,公式化简后为 $M\times K\times N(1/R+1/Q+2/R)$。从公式中可得出,在访存部件理想化、Cache 容量足够大的情况下,如果问题规模不变,即 M、N、K 为定值时,矩阵的子块越大,总的数据搬运量越小,程序访存压力越小,由于分块策略产生额外数据传输开销越小。

考虑到实际的访存结构及容量,如果每次内层核心循环的计算数据都在 DCache 中,访存指令的性能最高,同时可以采用软件调度的方法掩盖访存延迟。为此假设参与的运算的矩阵块都存在于 DCache 中。循环体在完成子矩阵 $\boldsymbol{B}_i$,$\boldsymbol{B}_j$ 的复制工作后,进行矩阵乘法计算并更新 $\boldsymbol{C}_{ij}$的值的时候,相关数据在运算时不发生 Cache 替换。即 $\boldsymbol{A}_i+\boldsymbol{B}_j+\boldsymbol{Sa}+\boldsymbol{Sb}+\boldsymbol{C}_{ij}$ 的总大小小于一级 Dcache 的容量(64kB),公式如下:

$$(2\times(P\times Q+Q\times R)+P\times R)\times 8<64\times 1024 \tag{1}$$

矩阵分块后如果数据均保存在 Dcache 中,那么矩阵的分块较小,根据数据搬运量公式,数据的复制时间会明显延长。核心计算部分性能提升,但是整个测试程序的性能下降。

在 Goto BLAS 算法中,矩阵 $\boldsymbol{A}$ 按列主序进行存储。中层循环开始时子矩阵 $\boldsymbol{A}_i$ 数据复制到 $\boldsymbol{Sa}$ 中时,数据地址跨度为 $M\times Q\times 8$ 字节。读入的子矩阵 $\boldsymbol{A}_i$ 在 Cache 中的数据以 $P\times 8$ 字节为一组,每组数据间隔$(M-P)\times 8$ 字节。为了使这些数据读入时不被替换出 Cache,避免计算时重新载入,数据容量要小于 Cache。在龙芯 3A2000 中,一级数据 Cache 共 4 路,每路 16kB,总容量为 64kB。16 路 VCache 容量为 256kB,要满足如下条件:

$$M\times Q\times 8<(64+256)\times 1024 \tag{2}$$

每个 Cache 行大小为 64 字节,$\boldsymbol{A}_i$ 中的子矩阵占用 $P\times Q/8$ 行。该矩阵中每行的 P 个数据占用连续的 Cache 行,但是矩阵中 Q 个数据块的地址不连续的。考虑到最差的情况,一级 Cache 中每路有$Q/4$个数据块,不同的数据块在 Cache 中的索引重复。那么数据块的间隔$(M\times 8)$满足

$$M\times 8=16\times 1024/(Q/4) \tag{3}$$

在底层的 Kernel 调用中要求参数 Q 取值不小于 64。那么 M 取值 128。当 M 增大时,数据复制过程中程序会反复替换 Cache 中的某些行,将 $\boldsymbol{Sa}$ 中的内容置换出 Cache。在矩阵分块大小为一路到两路 Cache 大小时,完成子矩阵复制后,缓冲区 $\boldsymbol{Sa}$ 的数据基本都在 Cache 中。在后继的优化算法中将矩阵分块大小限制在 Cache 容量的一半以内。

在最内层的迭代计算中,程序读取 $\boldsymbol{Sa}$ 和 $\boldsymbol{Sb}_j$ 和子矩阵 $\boldsymbol{C}_{ij}$。复制子矩阵 $\boldsymbol{B}_{kj}$时,每次复制的数据量大小为 $4\times Q$。计算完成后 $\boldsymbol{Sa}$ 和 $\boldsymbol{Sb}_j$ 在 Cache 中优

先级最高。每次计算完成后下次循环复制 $\boldsymbol{Sb}_{j+1}$ 和 $\boldsymbol{B}_{kj+1}$ 时不会替换掉 Cache 中的 ***Sa*** 和 $\boldsymbol{Sb}_j$ 的数据。缓冲区 ***Sb*** 根据分块的大小可能会被替换到下一级的 VCache 中。中层循环开始时,$\boldsymbol{A}$ 的子矩阵分块大小为一路或两路 Cache 行容量时,Cache 中的很多内容会被替换掉。复制工作完成后,缓冲区 ***Sa*** 和子矩阵 $\boldsymbol{A}_i$ 的内容存在 DCache 中,缓冲区 ***Sb*** 的一部分会被替换到 VCache 中。为了使 ***Sb*** 计算中不被替换出片内缓存,分块大小要满足

$$(2 \times (P \times Q) + Q \times R) \times 8 < 512 \times 1024 + 64 \times 1024 \tag{4}$$

在龙芯 3A2000 芯片中,为了防止 VCache 中的内容被 DCache 中替换次数过多,进而替换到速度更慢的 SCache 中。参与乘法运算的矩阵大小应小于 VCache 的容量。这样 $\boldsymbol{A}_i$ 子矩阵的内容不容易被替换出片内 Cache。

根据上面的分析,在分块方案中,矩阵 A 的分块最好选择 Cache 一路大小的整数倍,最大数值为一级数据 Cache 容量的一半。$\boldsymbol{B}$ 矩阵的大小为 VCache 与 DCache 容量的差值或者 VCache 与分块矩阵大小的差值。

3 Goto BLAS 库的优化方案

3.1 矩阵乘法核心的循环的优化

Goto BLAS 库中的热点 DGEMM 函数主要有 3 种操作,即矩阵数据复制、矩阵 $\boldsymbol{C}$ 与常数相乘以及子矩阵 $\boldsymbol{A}_i$ 和 $\boldsymbol{B}_j$ 的乘法运算,其中矩阵乘法部分最为耗时。在 Goto BLAS 库中,这些函数都采用平台相关的汇编语言实现相关功能,上层调用时根据当前运行的系统平台选择合适的实现方案。普通矩阵乘法采用 3 层循环完成,算法的核心操作为:

$$\boldsymbol{C}[i, j] = \sum_{n=0}^{k-1} \boldsymbol{C}[n, j] + \boldsymbol{A}[i, n] + \boldsymbol{B}[n, j] \tag{5}$$

针对龙芯 3A2000 处理器,根据其结构特点,采取的软件优化方法有:

(1) 采用高效的乘加指令。龙芯 3A2000 的处理器中配置了 2 个全流水的浮点功能部件。每个浮点功能部件支持三操作数的乘加操作。该操作可以减少计算指令条数,缩短整个程序的运行时间。

(2) 循环展开。循环展开是一种常用的软件编译优化方法。通过减少分支指令条数,由于分支指令在流水线中开销较大,该方式可以显著提高程序的运行性能。

(3) 指令调度及软件流水。软件流水通过调整循环体的操作方法,改变执行指令的序列,降低指令的数据相关性。减少因为数据相关导致的流水线停顿。

在乘法的内层循环中,将参与运算的 $\boldsymbol{A}[i, n]$ 及 $\boldsymbol{B}[n, j]$ 及相邻的数据 $\boldsymbol{B}[n, j+x-1]$,以及与 $\boldsymbol{B}[n, j]$ 运算相关的矩阵 $\boldsymbol{A}$ 的数据 $A[i+y-1, n]$ 同时取出来。此时核心循环的计算量为 $x \times y$,矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的访存操作量为 $x+y$。矩阵 $\boldsymbol{C}$ 的数据量为 $x \times y$,访存操作发生在第二层循环中。内层循环中忽略矩阵 $\boldsymbol{C}$ 的数据存取操作,计算需要使用的数据量大小为 $x \times y + x + y$。由于 MIPS 体系结构的逻辑寄存器数量为 32,同时考虑到 Cache 的访存延迟,在龙芯 3A2000 的处理器中,一级数据 Cache 延迟为 4 个时钟周期。Vcache 的访存延迟为 20 多个时钟周期。为了使功能部件全流水执行,算法将矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的数据提前一个循环体复制到寄存器中。一个循环体中的处理器需要保存本次循环参与计算的数据,同时预取下次迭代需要的矩阵数据即需要的寄存器数量,要满足:

$$x \times y + 2(x + y) < 32 \tag{6}$$

根据上节的瓶颈分析,子矩阵 C_{ij} 每次计算的列数要小于或等于一级数据 Cache 的路数,即 $x < 4$。根据浮点乘加指令 MADD 的延迟参数,每次循环计算指令的条数至少为 10。由于处理器中访存部件和浮点功能部件的处理能力差异,为了实现访存计算的平衡,避免访存能力不足导致计算功能部件效率达不到浮点峰值,循环体的计算指令数量应大于访存指令数目,即

$$x \times y \geqslant 10,\ x + y \leqslant x \times y \tag{7}$$

根据上述分析,设定 x 和 y 同时为 4。循环每次需要存储 16 个矩阵 $\boldsymbol{C}$ 的数据、4 个矩阵 $\boldsymbol{A}$ 的数据和 4 个矩阵 $\boldsymbol{B}$ 的数据,以及预读 8 个下次运算参与

的数据。循环中每次计算 16 次。由此,内层核心计算循环由 16 条计算指令和预取的 8 条访存指令组成。计算同一个矩阵 ***C*** 间隔 16 条指令。具体的实现方式见图 2。

```
ldc1 A0, 4 * offset __64(a _ offset)
ldc1 A1, 5 * offset _64(a _ offset)
  MADD _ D(C0, X0, Y0)
  MADD _ D(C1, X1, Y0)
ldc1 B0, 4 * offset _64(b _ offset)
ldc1 B1, 5 * offset _64(b _ offset)
  MADD _ D(C4, X0, Y1)
  MADD _ D(C5, X1, Y1)
ldc1 A2, 6 * offset _64(a _ offset)
ldc1 A3, 7 * offset _64(a _ offset)
  MADD _ D(C2, X2, Y0)
  MADD _ D(C6, X2, Y1)
ldc1 B2, 6 * offset _64(b _ offset)
ldc1 B3, 7 * offset _64(b _ offset)
  MADD _ D(C8, X0, Y2)
  MADD _ D(C9, X1, Y2)
  MADD _ D(C3, X3, Y0)
  MADD _ D(C7, X3, Y1)
  MADD _ D(C12, X0, Y3)
  MADD _ D(C13, X1, Y3)
  MADD _ D(C10, X2, Y2)
  MADD _ D(C11, X3, Y2)
  MADD _ D(C14, X2, Y3)
  MADD _ D(C15, X3, Y3)
```

图 2　核心循环的汇编指令实现

在程序的实际运行过程中,在循环结束和开始时还要进行额外的数据处理。包括矩阵 ***C*** 的数据访问。循环第一次计算需要读取矩阵 ***A*** 和 ***B***、浮点乘加之后的结果处理,以及分支指令处理带来的额外开销。这些共需要 30 到 50 个时钟周期。为了使核心计算的浮点部件效率在 90% 以上,那么排除额外的处理器开销,一次循环中浮点计算部件的时钟周期应该为额外开销的 10 倍左右。即每次循环的浮点计算时间为 500 个时钟周期左右。由于处理器配置了两个浮点功能部件,循环展开后的浮点计算指令条数为 1000 左右。核心循环体内的浮点指令数量为 16,那么循环展开的力度为 64。所以核心循环中计算矩阵的分块设置为 4×64 与 64×4。图 3 为核心算法中矩阵乘法的流程示意图。

3.2　数据预取

在核心循环中,缓冲区 ***Sa*** 在 DCache 中,缓冲区 ***Sb*** 根据分块策略可能会分布在 DCache 或者 VCache 中。在核心循环中对缓冲区 ***Sb*** 进行预取 $4\times Q$ 个数据。因为数据块连续且规模较大,采用软件预取指令 prefetch 实现该功能。对于子矩阵 ***C***,每次计算时访问 16 个数据,数据量小,不适合块式预取模式。采用 load 指令预加载数据到 0 号寄存器完成预取功能。由于矩阵 ***C*** 延迟大,在矩阵计算循环展开的下半部分预取下次计算所需的子矩阵数据。

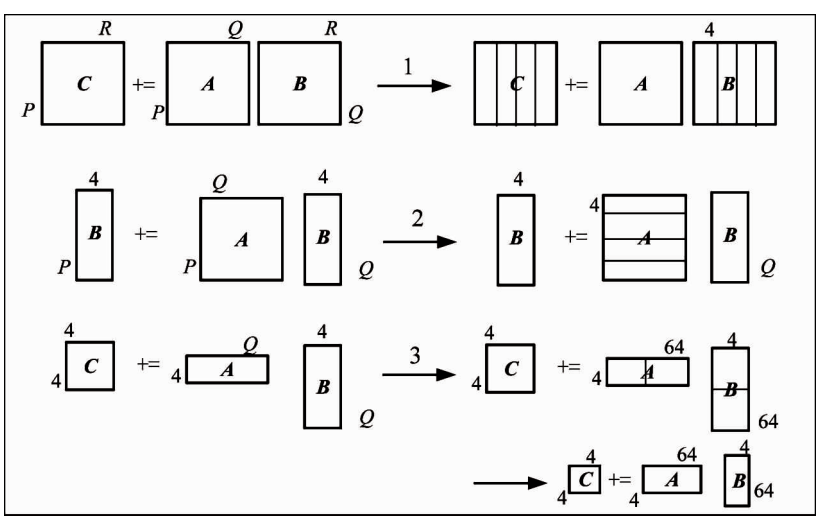

图 3　核心算法中矩阵乘法的流程示意图

3.3　矩阵分块的配置

根据上面章节，在 DGEMM 函数中，对于矩阵 ***A*** 分块大小为 DCache 的容量的一半左右，矩阵 ***B*** 的分块大小为 VCache 与 DCache 的差值，或者 VCache 与 ***A*** 分块大小的差值。由于核心循环的展开的粒度为 64，所以矩阵 ***A*** 子块的列数最好是 64 的倍数。同时矩阵的子块应该越大越好。

不考虑 TLB 失效的系统性能影响，在龙芯 3A2000 处理器系统中对不同的分块方式进行 DGEMM 函数的效率测试。图 4 实验中的系统采用 fix TLB 技术，TLB 项数配置为 1024，在程序运行期间几乎不发生 TLB 失效事件。在矩阵规模为 512 ×512 时，采用不同分块策略下的浮点部件核心计算效率。在矩阵分块大小配置为 32 ×128 和 128 ×128 时，核心循环的效率取得最大值，超过 88%，和理论分析的结果一致。

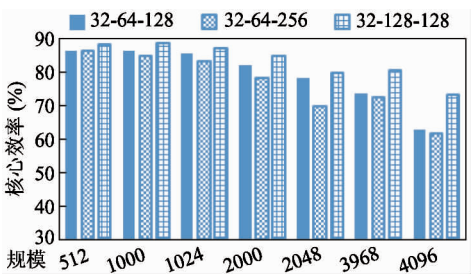

图 4　不同分块策略下的核心计算效率

4　实验平台和试验结果

在处理器设计的初期，我们采用 VCS[9]（Verilog Compiled Simulator）软件仿真器对处理器模型进行分析验证，采用 Verdi[10] 的波形调试工具分析功能部件的信号波形。基于该平台，我们根据 DGEMM 的处理流程，实现了矩阵分块乘法的程序。实验中统计浮点计算部件的 IPC 数值，在软件模拟环境中超过 1.8。由此得出浮点功能部件的效率可达到 90%。没有达到 100% 的原因有：

（1）在程序循环第一次执行的开头部分，处理器读取指令时，浮点部件中无指令不工作，循环体的第一次迭代工作效率低。

（2）每次循环间隔需要对相关的参与计算的数据进行处理，包括准备子矩阵 ***A*** 和 ***B*** 第一次计算需要的数据，存取子矩阵 ***C*** 的部分计算结果等。此时浮点单元中无执行指令，计算部件空闲。

在龙芯 3A2000 芯片流片后，在验证实验板上，我们配置处理器频率为 800MHz，DDR 工作频率 500MHz。在无相关的软硬件优化方法条件下，观察不同分块模式下矩阵的核心效率如图 4 所示。从图中数据可以分析出，矩阵规模增加后，矩阵乘法的效率开始降低，问题规模增大到 4096 后，效率下降明显。原因是计算过程中，矩阵 ***C*** 规模变大导致 DCache 中的数据被替换，子矩阵 ***A*** 或 ***B*** 被替换出一级 Cache，计算时发生 Cache 失效，程序性能下降。

另一个性能下降的原因是 TLB 失效次数随着矩阵 ***C*** 规模增加而增多。在每次分块乘法计算时，矩阵 ***C*** 占用页数为

$$page = R \times \min(1, M \times 8/page\_size) \qquad (8)$$

式中，$page$ 为页数，$page\_size$ 为页大小。本实验平台内核配置内存页大小 16kB。TLB 项数为 64，一个 TLB 表项存储两条页面信息。完整的矩阵乘法需要页数为 $N/R \times page$。TLB 采用随机替换算法，该算法在 TLB 表项占用超过一半时，TLB 缺失现象会大量发生。矩阵规模为 4096 时，矩阵 ***C*** 每次参与计算的 16 个数据在 Cache 中的索引相同，占用同一索引的 4 路 Cache 行，把 Cache 中原来缓冲区 ***Sa*** 和 ***Sb*** 的数据替换到 VCache 中。计算时访存延迟增加，浮点部件等待参与计算数据造成流水线停顿，性能下降。为了减少 TLB 失效次数，将 R 的数值降低。并使矩阵 ***A***，***B*** 分块规模小于 DCache，P、Q、R 分块参数为 $P = 32$，$Q = 64$，$R = 32$ 以及 $P = 32$，$Q = 128$，$R = 128$ 时，矩阵乘法的效率如图 5 所示。

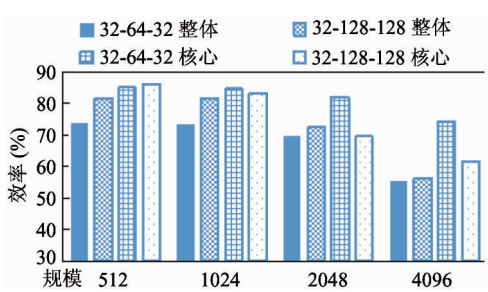

图 5　不同分块策略下的效率对比图

当矩阵分块变小时,分块数据可以全部存放在一级 DCache 中,数据读取延时最小。但是根据性能瓶颈分析小节,分块规模变小时,矩阵的数据搬运次数增加,复制时间随着增加,总体性能反而下降。

为了减少 TLB 失效的事件,提升程序运行的效率。龙芯 3A2000 处理器中扩大了 TLB 的容量,系统可以使用 1024 项 TLB。在图 6 的实验中,我们采用 1024 项 TLB(fix TLB)提升矩阵乘法的性能。结果表明在矩阵规模较小时,系统的 TLB 失效次数不多,运行性能对 TLB 配置不敏感。随着矩阵规模增加,未使用 fix TLB 技术的系统性能明显下降,使用该技术的系统效率无太大变化。

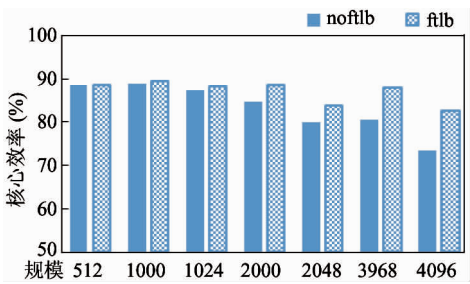

图 6　TLB 对矩阵乘法效率的影响

龙芯 3A2000 处理器提供多种数据预取技术提升应用程序的性能。在矩阵分块为 $P = 32$, $Q = 128$, $R = 128$ 的参数配置下,软硬件预取模式对核心效率的影响如图 7 所示。相对于普通运算模式,数据预取策略可以使核心乘法效率提高 10% 左右。核心循环的性能提升主要通过软件预取方法获得。硬件预取主要提升数据复制操作的性能。在运算中,数据搬运操作占不超过 10% 的乘法运行时间。所以硬件预取方法仅提升 1% 的浮点效率。

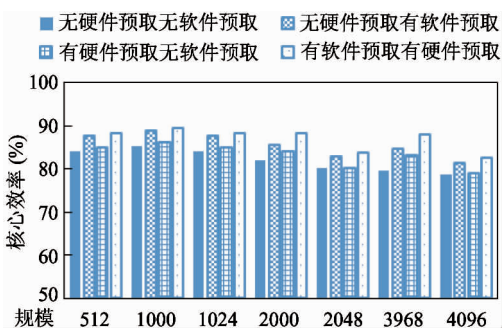

图 7　不同预取策略对应的核心循环效率

在进行完整的 Linpack 测试时,配置不同的矩阵规模,同时采用上述的软硬件优化方法。实验数据结果如图 8 所示。在矩阵规模较小时,采用单核心运行 Linpack 测试,优化后浮点性能可以达到 2.7Gflops 左右。根据处理器的结构组成,理论峰值性能为

$$4 \times 800\text{Mflops} = 3.2\text{Gflops} \tag{9}$$

优化后的 Linpack 的浮点效率为 84%。

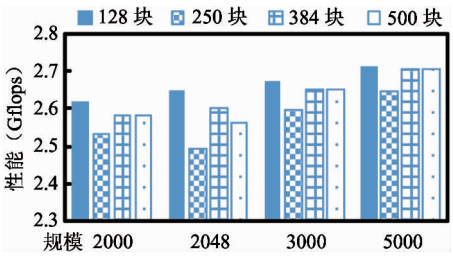

图 8　优化后的 Linpack 程序测试结果

当矩阵规模变大,Linpack 的问题规模 N 为 20000,分块大小取值 500 的情况下,Linpack 的实验结果可以到达 2.77Gflops。

5　结　论

科学计算的性能提升一直是处理器研究领域的热点问题。其中的数学库的优化调整是改善系统性能的重要工作。本文详细分析了 Goto BLAS 库的行为模式,结合国产处理器龙芯 3A2000 的结构特点,对核心算法进行了全面的优化改进。通过分析矩阵乘法的流程及数据访问方法,提出了合理的分块策略。针对算法核心循环的任务目标,重新改写了底层汇编代码。同时采用数据预取的策略,使运算部件的吞吐量与访存带宽相匹配。此外,本实验使用处理器的 fix TLB 技术,显著地减少系统的 TLB 失效次数。由于综合采用了上述多种优化方式,使得 Linpack 的核心循环效率可以达到 90%。优化效果显著。同时这些方法也可作为其他的程序优化方案的参考。本优化方法不局限于 MIPS 架构的处理器,也适用于其他的处理器平台。后续工作中,可以结合网络延迟参数、节点内部的数据传输方式、集群系统的结构组成等对云平台的计算模块做进一步的优化分析,提升整个计算系统的综合性能。

参考文献

[1] Dongarra J J, Piotr L, Antoine P. The LINPACK Benchmark: past, present and future. *Concurrency & Computation Practice & Experience*, 2003, 15(9):803-820

[2] Wang P, Turner G, Lauer D A, et al. LINPACK Performance on a Geographically Distributed Linux Cluster. In: Proceedings of International Parallel and Distributed Symposium, Santa Fe, USA, 2004. 245b

[3] Wu R Y, Wang W W, Wang H D, et al. Design of loongson gs464e processor architecture. *SCIENTIA SINICA Informationis*, 2015, 45 (4): 480-500

[4] MIPS Technologies Inc. MIPS64 Architecture for Programmers Volume I: Introduction to the MIPS64 Architecture. https://imgtec.com: Imagination, 2005

[5] Belter G, Jessup E R, Karlin I, et al. Automating the generation of composed linear algebra kernels. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, Bangalore, India, 2009. 1-12

[6] Wang L, Zhang Y, Zhang X, et al. Accelerating Linpack performance with mixed precision algorithm on CPU + GPGPU heterogeneous cluster. In: Proceedings of International Conference on Computer and Information Technology, Dhaka, Bangladesh, 2010. 1169-1174

[7] Wang F, Yang C, Bai J. Hybrid MPI/OpenMP optimization in Linpack benchmark on multi-core platforms. In: Proceedings of the International Conference on Computer Science & Education, Colombo, Sri Lanka, 2013. 917-920

[8] 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(01):163-167

[9] Synopsys. VCS. http://www.eve-team.com: Synopsys, 2010

[10] Synopsys. Verdi. https://www.synopsys.com: Synopsys, 2011

Implementation of a high-performance Goto BLAS based on Loongson 3A2000 processor

Zhang Hualiang[*,**,***], Huang Qiyin[*,**], Wu Shaoxiao[*,**,***]

([*] Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

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

([***] Loongson Technology Corporation Limited, Beijing 100190)

Abstract

Linpack was applied to evaluation of the performance of a computer system, and the Goto BLAS library was used as the function operation library. The performance of the library has a large impact on Linpack test results. To achieve its high performance, the study observed the performance expression of the Goto BLAS library on the Loongson 3A2000 processor, and analyzed the testing software's execution flow and data processing methods, and then, according to the structural features of the processor, reasonably allocated the block matrix and optimized the scheme for implementation of the core loop in the function. Meanwhile, the data-fetching technologies of software and hardware, and the optimized TLB configuration schemes were adopted. With the combined effects of these optimizations, the efficiency of float point component on the simulation platform reached more than 90%, which means the optimization schemes achieved the significant results in this experiment.

Key words: Goto BLAS, performance optimization, Linpack, matrix operations, data prefetching