doi:10.11887/j.cn.201406002 http://journal.nudt.edu.cn

支持原位计算的高效三角矩阵乘法向量化方法*

刘 仲,田 希,陈 磊
(国防科技大学 计算机学院,湖南 长沙 410073)

摘 要:向量化算法映射是向量处理器的难点问题。提出一种高效的支持原位计算的三角矩阵乘法向量化方法:将L1D配置为SRAM模式,用双缓冲的乒乓方式平滑多级存储结构的数据传输,使得内核的计算与DMA数据搬移完全重叠,让内核始终以峰值速度运行,从而取得最佳的计算效率;将不规则的三角矩阵乘法计算均衡分布到各个向量处理单元,充分开发向量处理器的多级并行性;将结果矩阵保存在乘数矩阵中,实现原位计算,节省了存储空间。实验结果表明,提出的向量化方法使三角矩阵乘法性能达到1053.7GFLOPS,效率为91.47%。

关键词:三角矩阵乘法;原位计算;向量化;向量处理器
中图分类号:TP391.4 **文献标志码**:A **文章编号**:1001-2486(2014)06-007-05

Efficient vectorization method of triangular matrix multiplication supporting in-place calculation

LIU Zhong, *TIAN Xi*, *CHEN Lei*
(College of Computer, National University of Defense Technology, Changsha 410073, China)

Abstract: The vectorization of algorithm mapping for vector processors is a critical issue. An efficient vectorization method of triangular matrix multiplication which supports the in-place calculation was presented. L1D was configured as SRAM and the ping-pong pattern with double buffering was designed to smooth the data transfers of multilevel storage structure, which made the kernel computation overlap the DMA data transfer fully and run with peak speed throughout, so then the optimal computation efficiency was achieved. Irregular triangular matrix multiplication computation was evenly distributed to all vector processing elements to fully exploit multiple levels of parallelism for vector processor. Result matrix was stored in multiplier matrix, thus, the in-place calculation was achieved and the memory space was saved. Experimental results show that the performance of triangular matrix multiplication attained from the presented vectorization method achieves 1053.7 GFLOPS and the efficiency of that reaches to 91.47%.

Key words: triangular matrix multiplication; in-place calculation; vectorization; vector processor

随着大型稠密线性方程组求解、雷达信号处理、高清视频和数字图像处理等计算密集型应用对高性能计算的需求日益增长,计算机体系结构出现显著变化,出现许多新型体系结构,向量处理器体系结构是其中之一。向量处理部件包括若干向量处理单元,每个处理单元上包含丰富的运算部件,具有强大的计算能力,能够大幅度提高系统的计算性能,但同时对软件开发提出了新的挑战,如何针对向量处理器多处理单元、多功能部件等体系结构特点,充分开发各个层次的并行性,将各类应用高效地向量化是当前面临的主要困难[1-2]。

基本线性代数函数库(Basic Linear Algebra Subprograms,BLAS)是各种科学计算所必需的核心数学库,广泛应用于物理学、电子工程学、生物学、经济学、计算科学等科学与工程计算。针对不同体系结构的BLAS优化一直是国内外的研究热点。Intel针对自己的CPU实现了专门优化的基本数学运算库MKL[3];Goto等针对不同体系结构采用手工汇编优化实现了高效的GotoBLAS库[4-5],采用自适应优化技术实现了ATLAS库[6]等;Volkov等研究了面向GPUs的BLAS库的优化实现[7];Marker等研究了面向多线程体系结构平台上的矩阵乘法的优化实现[8];张先轶等面向龙芯3号多核处理器开发了高性能的OpenBLAS库[9]。三角矩阵与矩阵乘法和三角矩阵求解两类函数库是BLAS库中最常被调用的函数库之一。

三角矩阵乘法是典型的计算密集和访存密集型运算,对处理器的运算能力、访存带宽及延迟的

* **收稿日期**:2014-04-22
基金项目:国家自然科学基金资助项目(61133007)
作者简介:刘仲(1971—),男,湖南邵东人,副研究员,博士,E-mail:zhongliu@nudt.edu.cn

要求非常高,优化三角矩阵乘法,对提高程序的运行速度、发挥处理器的运算能力具有重要的意义。与普通矩阵相比,三角矩阵有一半的矩阵元素为0,在计算和存储上套用普通的矩阵乘法不能有效提升三角矩阵的计算性能。并且,在面向向量处理器平台上的计算时,一方面三角矩阵不像普通矩阵是方阵,难以实现规整化的数据对齐,不能平衡各个处理单元的计算负载;另一方面,向量数据访问单元支持的向量数据 Load/Store 一般只支持按行读取,不支持按列读取。

1　向量处理器 Matrix 的体系结构

Matrix 是一款面向高密度计算应用、基于共享存储结构的高性能多核向量处理器。单核结构如图 1 所示,每个单核是独立的超长指令字(Very Long Instruction Word, VLIW)体系结构,包括标量处理部件(Scalar Processing Unit, SPU)和向量处理部件(Vector Processing Unit, VPU),SPU 负责标量任务计算和流控,SPU 和 VPU 可通过共享寄存器交换数据。Matrix 每时钟周期发射 11 条指令,包括 5 条标量指令和 6 条向量指令。指令派发单元对执行包进行识别,并将其中的指令派发到相应的功能单元中执行。VPU 负责向量计算,包括 16 个向量处理单元(Vector Processing Element, VPE),每个 VPE 含 1 个局部寄存器文件,以及 3 个浮点累积乘单元(Floating-Point Multiply-Accumulator, FMAC)、1 个 BP 和 2 个 Load/Store 共 6 个并行功能部件。局部寄存器文件包含 64 个 64 位寄存器,所有 VPE 的同一编号的局部寄存器在逻辑上又组成一个 1024 位的向量寄存器。向量数据访问单元支持向量数据的 Load/Store,提供阵列存储器(Array Memory, AM),每周期同时支持 2 个 Load/Store 指令。多核采用共享 DDR 的存储结构,DDR 与 L1D 间支持多核共享的全局缓存(Global Cache, GC),方便数据共享。

2　三角矩阵乘法的向量化方法

对于普通的矩阵乘法 $\boldsymbol{C}=\boldsymbol{A}\times\boldsymbol{B}$,其中 $\boldsymbol{A}$ 为 $m\times n$阶矩阵,$\boldsymbol{B}$ 为 $n\times k$ 阶矩阵,$\boldsymbol{C}$ 为 $m\times k$ 阶矩阵。结果矩阵 $\boldsymbol{C}$ 按照式(1)计算:

$$\boldsymbol{C}_{ij}=\sum_{r=0}^{n-1}\boldsymbol{A}_{ir}\times\boldsymbol{B}_{rj} \quad (1)$$

$$(i=0,\cdots,m-1,\ j=0,\cdots,k-1)$$

根据式(1)计算结果矩阵 $\boldsymbol{C}$,在计算时间方面,每个 $\boldsymbol{C}$ 矩阵元素的计算需要 n 次乘法和 $n-1$ 次加法,完成矩阵 $\boldsymbol{C}$ 的计算需要 $m\times k\times n$ 次乘法和 $m\times k\times(n-1)$ 次加法。在存储方面,若以矩阵元素为一个存储单位,则需要存储空间为 $(m\times n+n\times k+m\times k)$ 单位。三角矩阵有一半的矩阵元素为0,简单地套用矩阵乘法实现三角矩阵乘法,在计算效率和存储效率方面都是低效的,因此,需要针对三角矩阵的计算和存储特点,以及所在处理器平台的体系结构特点制定更高效的计算方法。

2.1　多核并行三角矩阵乘法算法

假定三角矩阵乘法计算:$\boldsymbol{C}=\boldsymbol{TB}$,其中 $\boldsymbol{T}$ 为上三角矩阵或下三角矩阵,$\boldsymbol{B}$ 为普通矩阵,$\boldsymbol{C}$ 为结果矩阵。根据多核的共享存储结构特点,多核的三角矩阵乘法采用分块的思想实现并行。如图 2(a)所示,设处理器有 v 个核,标号为 $0,1,\cdots,v-1$。将 $\boldsymbol{B}$ 矩阵按列划分为 v 块,依次记为 $\boldsymbol{B}_0,\boldsymbol{B}_1,\cdots,\boldsymbol{B}_{v-1}$,分别由标号为 $0,1,\cdots,v-1$ 的处理

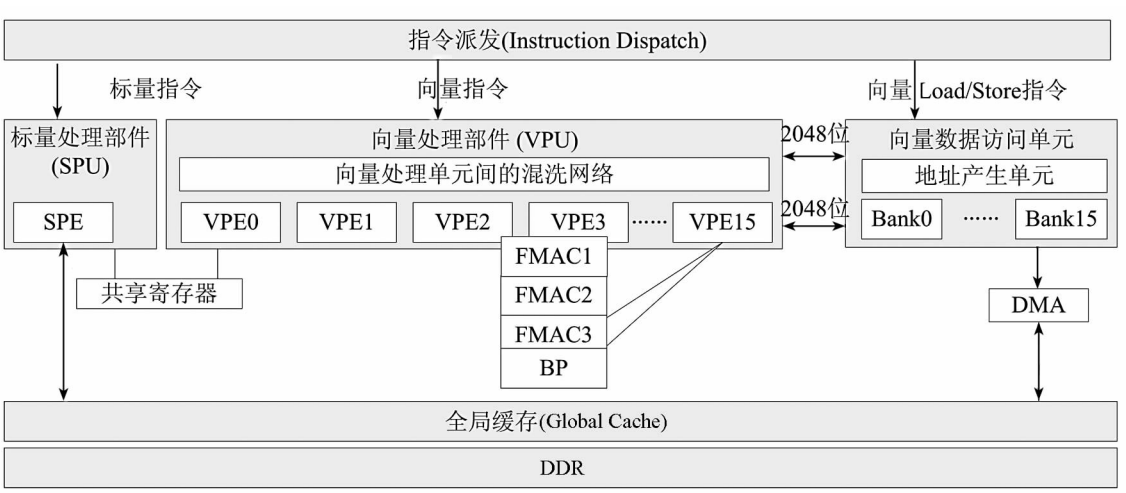

图 1　Matrix 的体系结构

Fig. 1　Architecture of Matrix

器核计算,结果矩阵 $\boldsymbol{C}$ 也按列进行相应的划分,则有:

$$(\boldsymbol{C}_0,\boldsymbol{C}_1,\cdots,\boldsymbol{C}_{v-1})=\boldsymbol{T}(\boldsymbol{B}_0,\boldsymbol{B}_1,\cdots,\boldsymbol{B}_{v-1})$$

如图 2(b)所示,其中 $\boldsymbol{C}_i=\boldsymbol{T}\boldsymbol{B}_i$ 由相应的核 i 完成计算。

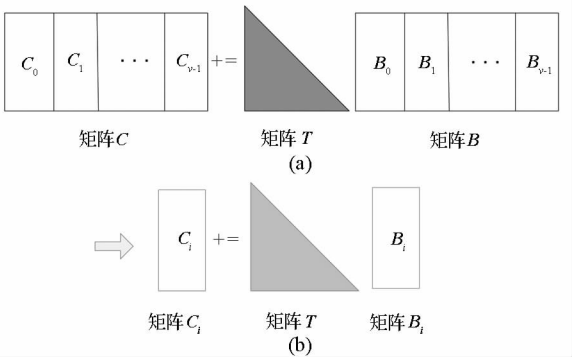

图 2　并行三角矩阵乘法方法

Fig. 2　Parallel triangular matrix multiplication

2.2　单核三角矩阵乘法向量化方法

提高向量处理器计算效率的一个关键技术是解决“存储墙”问题。高性能计算中,必须平滑各级存储之间的数据搬移,保证处理器计算所需要的数据供应,才能取得高效的处理器计算效率。Matrix 处理器包括寄存器文件、L1D、AM、GC 和 DDR 多级存储结构,其中 L1D 可配置为全 Cache 和全 SRAM 两种模式。传统的矩阵计算采用分块矩阵来尽可能地提高 Cache 的命中率,从而提高处理器的计算效率,但总是存在 Cache 不命中的情况,存储等待导致处理器停顿,因而很难取得更高的计算效率。本文提出的方法是采用 SRAM 方式,将 L1D 配置为全 SRAM 模式,并采用双缓冲的乒乓方式平滑多级存储结构的数据传输,使得内核的计算与 DMA 数据搬移完全重叠,让内核始终处于峰值运行状态,从而取得最佳的计算效率。

单核的三角矩阵乘法向量化方法如下:

1)将被乘数三角矩阵 $\boldsymbol{T}$ 中的三角矩阵元素按行连续存储;设被乘数三角矩阵 $\boldsymbol{T}$ 为 $m\times m$ 阶矩阵,乘数矩阵 $\boldsymbol{B}_i$ 为 $m\times k$ 阶矩阵,计算三角矩阵 $\boldsymbol{T}$ 与乘数矩阵 $\boldsymbol{B}_i$ 的乘法:$\boldsymbol{C}_i=\boldsymbol{T}\times\boldsymbol{B}_i$,结果矩阵 $\boldsymbol{C}_i$ 为 $m\times k$ 阶矩阵;

2)根据向量处理器的向量处理单元个数和向量处理单元的 FMAC 部件个数对乘数矩阵 $\boldsymbol{B}_i$ 按列划分为若干个子矩阵 $\boldsymbol{B}_{ij}$;具体划分方法是,设向量处理器的向量处理单元个数为 p,向量处理单元的 FMAC 部件个数为 q;对乘数矩阵 $\boldsymbol{B}_i$ 按列划分为子矩阵,子矩阵的行数与 $\boldsymbol{B}_i$ 矩阵一致,均为 m,子矩阵的列数固定为 $p\times q$,若 k 不是 $(p\times q)$ 的整数倍,则最后一个子矩阵的列数为 k 除以 $(p\times q)$ 的余数;记子矩阵的个数为 s,子矩阵依次记为 $\boldsymbol{B}_{i0}$, $\boldsymbol{B}_{i1}$, …, $\boldsymbol{B}_{is-1}$;采用分块算法实现结果矩阵 $\boldsymbol{C}$ 的计算:$(\boldsymbol{C}_{i0},\boldsymbol{C}_{i1},\cdots,\boldsymbol{C}_{is-1})=T\times(\boldsymbol{B}_{i0},\boldsymbol{B}_{i1},\cdots,\boldsymbol{B}_{is-1})$,令 $j=0$。

3)依次实现被乘数三角矩阵 $\boldsymbol{T}$ 与子矩阵 $\boldsymbol{B}_{ij}$ 的乘法,计算结果存储在原子矩阵 $\boldsymbol{B}_{ij}$ 的存储位置;

4)遍历完乘数矩阵 $\boldsymbol{B}_i$ 的全部子矩阵 $\boldsymbol{B}_{ij}$。判断是否还有未计算的子矩阵 $\boldsymbol{B}_{ij}$,若有,更新 $j=j+1$,转步骤 3;若无,则执行步骤 5;

5)单核的三角矩阵乘法的计算完成。

其中步骤 3 采用双缓冲的乒乓方式实现三角矩阵 $\boldsymbol{T}$ 与各子矩阵 $\boldsymbol{B}_{ij}$ 的乘法计算。

如图 3 所示,T 矩阵的数据由标量 L/S 部件从 L1D(配置为全 SRAM)中加载,$\boldsymbol{B}_i$ 和 $\boldsymbol{C}_i$ 矩阵的数据由向量 L/S 部件从 AM 中加载。SRAM 和 AM 都划分为 2 个缓冲区。图 3 中 1 至 4 栏用以说明三角矩阵乘法的核心计算与 DMA 数据传输重叠过程。

在第 1 栏中,准备第一个 AM 缓冲区和第一个 SRAM 缓冲区的数据;

在第 2 栏中,核心读取第一个 AM 缓冲区和第一个 SRAM 缓冲区的数据进行计算,同时启动 DMA 在后台搬移第二个 AM 缓冲区和第二个 SRAM 缓冲区的数据。在一个 AM 缓冲区计算期间,两个 SRAM 缓冲区计算和数据搬移重叠;

在第 3 栏中,核心读取第二个 AM 缓冲区的计算,同时启动 DMA 在后台搬移第一个 AM 缓冲区的计算结果,期间,两个 SRAM 缓冲区的计算和数据搬移重叠;

在第 4 栏中,核心读取继续第二个 AM 缓冲区的计算,同时 DMA 在后台搬移第一个 AM 缓冲区的数据,期间,两个 SRAM 缓冲区的计算和数据搬移重叠。

核心循环重复上述过程,直到计算结束。

2.3　支持原位计算的内核级三角矩阵乘法向量化方法

提高向量处理器计算效率的另一个关键技术是充分开发多核向量处理器的多级并行性。Matrix 这款向量处理器是 11 流出的 VLIW 结构,包含 16 个 VPE,VPE 支持 3 个并行的 FMAC 部件。为高效实现上一节算法步骤 3 中的乘数三角矩阵 $\boldsymbol{T}$ 与子矩阵 $\boldsymbol{B}_{ij}$ 的乘法计算,本节提出的下述内核级三角矩阵乘法向量化方法,能够充分开发向量处理器的部件级和指令级并行性,通过软件流水取得最佳的 FMAC 部件利用率,从而取得接

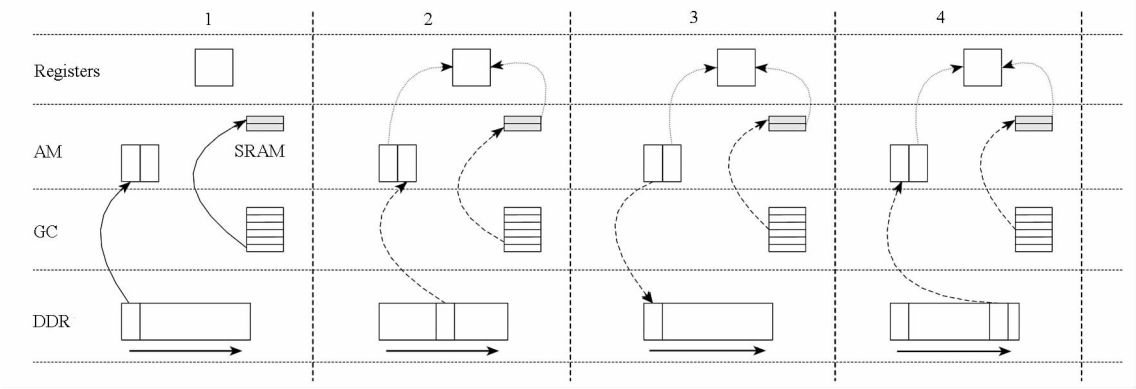

图 3　三角矩阵乘法计算与 DMA 数据传输重叠

Fig. 3　Overlap between triangular matrix multiplication computation and DMA data transfer

近峰值性能的计算效率。

如图 4 所示,内核级三角矩阵乘法向量化方法如下:

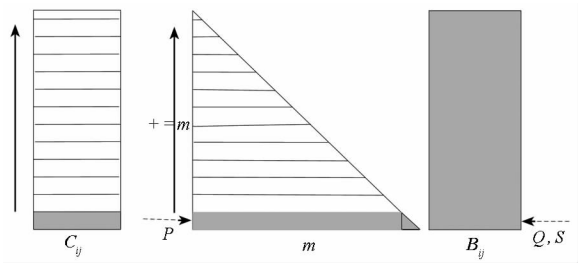

图 4　内核级三角矩阵乘法向量化方法

Fig. 4　Vectorization of kernel level triangular matrix multiplication

1)若三角矩阵 $\boldsymbol{T}$ 为上三角矩阵,则 P 指向三角矩阵 $\boldsymbol{T}$ 的第一行,Q 和 S 分别指向子矩阵 $\boldsymbol{B}_{ij}$ 的第一行;若三角矩阵 $\boldsymbol{T}$ 为下三角矩阵,则 P 指向三角矩阵 $\boldsymbol{T}$ 的最后一行,Q 和 S 分别指向子矩阵 $\boldsymbol{B}_{ij}$ 的最后一行;

2)向量处理器的 SPU 用标量存取指令读取三角矩阵 $\boldsymbol{T}$ 的第 P 行中的连续 q 个元素到 q 个标量寄存器,并用广播指令分别广播到 VPU 的 q 个向量寄存器;

3)向量处理器的 VPU 用向量存取指令读取子矩阵 $\boldsymbol{B}_{ij}$ 的第 S 行的 $q \times p$ 个元素到与步骤 2 不同的 q 个向量寄存器;

4)向量处理器的 VPU 对步骤 2 的 q 个向量寄存器和步骤 3 的 q 个向量寄存器分别执行乘法,乘法结果分别累加到不同的 q 个向量寄存器;

5)判断子矩阵 $\boldsymbol{B}_{ij}$ 中是否还有另一行数据,若有,则子矩阵 $\boldsymbol{B}_{ij}$ 中更新 S 为 S 的下一行(三角矩阵 $\boldsymbol{T}$ 为上三角矩阵的情况)或上一行(三角矩阵 $\boldsymbol{T}$ 为下三角矩阵的情况),转步骤 2;

6)将步骤 4 计算结果中的 q 个向量寄存器保存到子矩阵 $\boldsymbol{B}_{ij}$ 的第 Q 行;

7)判断三角矩阵 $\boldsymbol{T}$ 中是否还有另一行数据,若有,则三角矩阵 $\boldsymbol{T}$ 中更新 P 为 P 的下一行(三角矩阵 $\boldsymbol{T}$ 为上三角矩阵的情况)或上一行(三角矩阵 $\boldsymbol{T}$ 为下三角矩阵的情况);子矩阵 $\boldsymbol{B}_{ij}$ 中更新 Q 为 Q 的下一行(三角矩阵 $\boldsymbol{T}$ 为上三角矩阵的情况)或上一行(三角矩阵 $\boldsymbol{T}$ 为下三角矩阵的情况),子矩阵 $\boldsymbol{B}_{ij}$ 中更新 S 为 Q 行,转步骤 2;

8)被乘数三角矩阵 $\boldsymbol{T}$ 与子矩阵 $\boldsymbol{B}_{ij}$ 的乘法计算完成。

与传统的矩阵乘法计算相比,本文提出的三角矩阵乘法向量化方法能够大幅度提高三角矩阵乘法的计算效率,具有以下显著优点:

1) L1D 采用 SRAM 模式,用 DMA 双缓冲平滑多级存储结构的数据传输,使得内核的计算与 DMA 数据搬移完全重叠,让内核始终以峰值速度运行;

2) 充分利用三角矩阵中有一半元素为 0,而 0 元素不需要与矩阵 $\boldsymbol{B}$ 中的对应行的数据相乘,这样的方法能够大幅减少计算量;

3) 被乘数矩阵 $\boldsymbol{T}$ 中的三角矩阵元素按行连续存储,不参与计算的 0 元素不需要存储;实现了原位计算,结果矩阵是保存在乘数矩阵 $\boldsymbol{B}$ 中,节省了存储空间;

4) 每次向量计算都能够平衡各个处理单元的计算负载,充分发挥向量处理器的各个处理单元的计算能力;

5) 避免了通常向量处理器不支持矩阵的列向量数据的访问和向量处理单元之间的浮点归约求和(浮点归约求和的硬件开销很大)。

3　性能测试

在向量处理器 Matrix 上对不同三角矩阵规模

的三角矩阵乘法性能进行了测试(称 MatrixTRMM),并与 Intel CPU 平台上的 MKL10.2.3、NVIDIA GPU 平台上的 cuBLAS4.1[10] 和 DSP 平台的 TI C6678[11] 几种典型平台的算法库性能进行了比较。

图 5 显示了不同三角矩阵规模在单核 Matrix 上取得的计算性能和效率。其中横坐标是被乘数三角矩阵 $\boldsymbol{T}$ 的规模($m=n$),乘数矩阵 $\boldsymbol{B}$ 的列根据 2.2 小节中的算法确定的参数 48。从图中可以看出,随着三角矩阵的规模逐步增大,单核上的三角矩阵乘法取得的性能和效率不断提高,在 $m=2048$ 时,性能为 95.37GFLOPS,接近单核 Matrix 的峰值性能 96GFLOPS,效率为 99.35%。

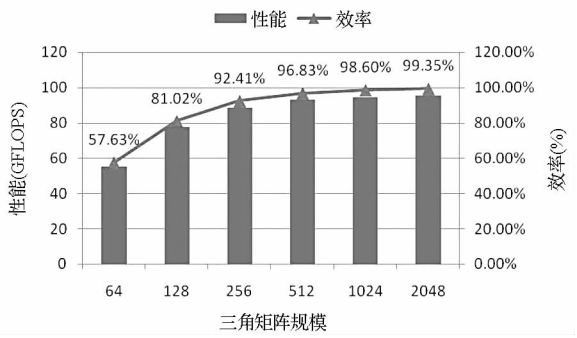

图 5　单核 Matrix 上三角矩阵乘法的计算性能和效率

Fig. 5　Triangular matrix multiplication performance and efficiency on single-core Matrix

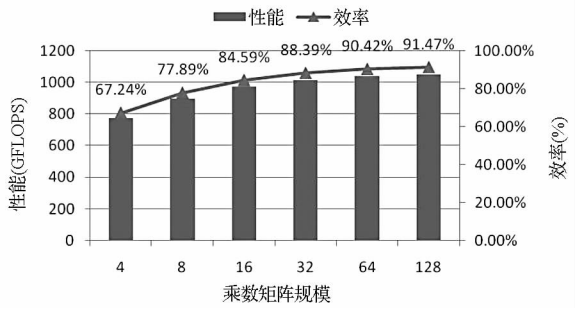

图 6　多核 Matrix 上三角矩阵乘法的计算性能和效率

Fig. 6　Triangular matrix multiplication performance and efficiency on multi-core Matrix

进一步测试以单核核心代码为基础开发的多核并行 TRMM 程序的性能。如 2.1 节算法所示,并行 TRMM 采用分块方法,为方便测试,设定其中的 $\boldsymbol{T}$ 规模为 512×512,$\boldsymbol{B}$ 为 $512\times48\times q$,图 6 显示了不同乘数矩阵规模(q 值)在多核 Matrix 上取得的计算性能和效率。从图中可以看出,随着乘数矩阵的规模逐步增大,多核并行 TRMM 程序取得的性能和效率不断提高,在 $q=128$ 时,性能为 1053.7GFLOPS,接近多核 Matrix 的峰值性能 1152GFLOPS,效率为 91.47%。与单核效率相比,多核并行 TRMM 程序的效率下降了约 8 个百分点,这是因为并行 TRMM 程序需要多核同步开销从而增加了总的执行时间,使得多核的效率降低了,但总的效率还是较高的。

图 7 给出了相同规模下(4k × 4k)下的 Matrix、GPU、CPU 和 DSP 上的双精度浮点三角矩阵乘法分别取得的性能和效率比较,其中 MatrixTRMM、cuBLAS、MKL 和 TI_BLAS 分别表示在 Matrix(12 核,双精度浮点峰值 1152GFLOPS)、Tesla C2090 (512 核,双精度浮点峰值 665GFLOPS)、Intel Xeon x5680(6 核,双精度浮点峰值 80GFLOPS)和 TI C6678(8 核,双精度浮点峰值 32GFLOPS)平台上报告的双精度浮点三角矩阵乘法计算取得的性能和效率。从图中可以看出,MatrixTRMM 取得的性能和效率显著高于其他算法库。

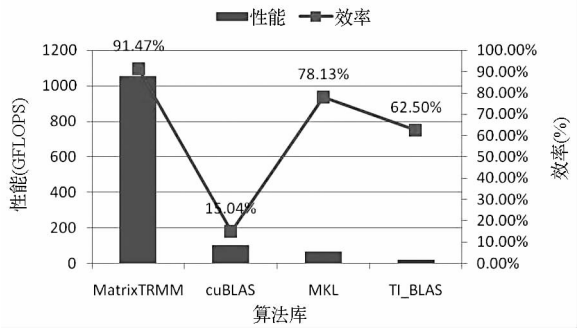

图 7　与其他处理器的性能和效率比较

Fig. 7　Performance and efficiency comparison with other processors

4　结论

传统的基于 Cache 的分块矩阵方法中,由于总是存在 Cache 不命中的情况,存储等待导致处理器停顿,使得很难以取得更高的计算效率。本文提出一种高效的支持原位计算的三角矩阵乘法向量化方法,采用 L1D 配置的 SRAM 模式,用双缓冲的乒乓方式平滑多级存储结构的数据传输,使得内核的计算与 DMA 数据搬移完全重叠,让内核始终以峰值速度运行,从而取得最佳的计算效率。另外,根据三角矩阵乘法特点,将不规则的三角矩阵乘法计算均衡分布到各个向量处理单元,以充分开发向量处理器的多级并行性。实验结果表明,提出的向量化方法使三角矩阵乘法取得了非常高效的计算效率。

(下转第 47 页)

－34.
WANG Xiaoqiang, ZHU Xi, MEI Zhiyuan, et al. Ballistic performances of ultra-high molecular weight polyethylene fiber-reinforced thick laminated plates[J]. Explosion and Shock Waves, 2009(1): 29－34. (in Chinese)

[8] Li V C, Mishra D K, Wu H C. Matrix design for pseudo strain-hardening fiber reinforced cementitious composites[J]. Materials and Structures, 1995, 28(10):586－595.

[9] Kamada T, Li V C. The effects of surface preparation on the fracture behavior of ECC/concrete repair system[J]. Cement and Concrete Composites, 2000, 22(6):423－431.

[10] Fisher G. Deformation behavior of reinforced ECC flexural members under reversed cyclic loading conditions[D]. USA: University of Michigan, 2002.

[11] 董祥, 高建明,吉伯海. 纤维增强高性能轻骨料混凝土的力学性能研究[J]. 工业建筑, 2005, 35(z1): 662,680－683.
DONG Xiang, GAO Jianming, JI Bohai. Study on mechanical properties of fiber reinforced high performance lightweight aggregate concrete[J]. Industrial Construction, 2005, 35(z1): 662,680－683. (in Chinese)

[12] Máca P, Sovják R, Konvalinka P. Mix design of UHPFRC and its response to projectile impact[J]. International Journal of Impact Engineering, 2014, 63:158－163.

[13] 中华人民共和国住房和城乡建设部,国家质量监督检验检疫总局. GB/T50081－2002 普通混凝土力学性能试验方法标准[S]. 北京: 中国建筑工业出版社, 2003.
Ministry of Housing and Urban-Rural Development of the People's Republic of China, General Administration of Quality Supervision, Inspection and Quaantine of People's Republic of China. GB/T50081－2002 Standard for test method of mechanical properties on ordinary concrete[S]. Beijing: China Architecture &Building Press, 2003. (in Chinese)

[14] 中华人民共和国住房和城乡建设部. JGJ/T221－2010 纤维混凝土应用技术规程[S]. 北京: 中国建筑工业出版社, 2010.
Ministry of Housing and Urban-Rural Development of the People's Republic of China. JGJ/T221－2010 Technical specification for application of fiber reinforced concrete[S]. Beijing: China Architecture &Building Press, 2010. (in Chinese)

(上接第 11 页)

参考文献(References)

[1] 刘仲,陈跃跃,陈海燕. 支持任意系数长度和数据类型的 FIR 滤波器向量化方法[J]. 电子学报, 2013, 41(2): 346－351.
LIU Zhong, CHEN Yueyue, CHEN Haiyan. A vectorization of FIR filter supporting any length and data types of coefficients[J]. Acta Electronics Sinica, 2013, 41(2):346－351. (in Chinese)

[2] 刘仲,邢彬朝,陈跃跃. 一种面向多核处理器的高效并行 PCA-SIFT 算法[J]. 国防科技大学学报, 2012, 34(4):103－107.
LIU Zhong, XING Binchao, CHEN Yueyue. An efficient parallel PCA-SIFT algorithm for multi-core processor[J]. Journal of National University of Defense Technology, 2012, 34(4):103－107. (in Chinese)

[3] Intel MKL Homepage[EB/OL]. [2014－04－24]. http://software.intel.com/en－us/articles/intel－mkl/.

[4] GotoBLAS Homepage. [EB/OL]. [2014－04－24]. http://www.tacc.utexas.edu/tacc－projects/gotoblas2.

[5] Goto K, van de Geijn R A. High-performance implementation of the level-3 BLAS[J]. ACM Transactions on Mathematical Software, 2008, 35(1):1－14.

[6] ATLAS Homepage. [EB/OL]. [2014－04－24]. http://math-atlas.SourceForge.net/.

[7] Volkov V, Demmel J W. Benchmarking GPUs to tune dense linear algebra[C]//Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, NY: IEEE Press, 2008: 131－142.

[8] Marker B, van Zee F G, Goto K, et al. Toward scalable matrix multiply on multithreaded architectures[C]//Proceedings of the 13th International European Conference on Parallel and Distributed Computing, Rennes, France, 2007:748－757.

[9] 张先轶, 王茜, 张云泉. OpenBLAS:龙芯 3A CPU 的高性能 BLAS 库[J]. 软件学报, 2011, 22(zk2):208－216.
ZHANG Xianyi, WANG Qian, ZHANG Yunquan. OpenBLAS: a high performance BLAS library on loongson 3A CPU[J]. Journal of Software, 2011, 22(zk2):208－216. (in Chinese)

[10] CUDA Programming Guide. [EB/OL]. [2014－04－24]. https://developer.nvidia.com/cublas.

[11] Ali M, Stotzer E, Igual F D, et al. Level-3 BLAS on the TI C6678 multi-core DSP[C]//Proceedings of IEEE 24th International Symposium on Computer Architecture and High Performance Computing, NY: IEEE Press, 2012:179－186.