# 面向多核向量处理器的矩阵乘法向量化方法

刘 仲 田 希

(国防科技大学计算机学院 长沙 410073)

**摘 要** 稠密矩阵乘法是大规模科学计算中许多算法的核心计算之一,文中提出一种高效的面向多核向量处理器的矩阵乘法向量化方法.提出一种按行计算的矩阵乘法向量化方法,该向量化方法的基本思想是每次同时计算 $\boldsymbol{C}$ 矩阵的一行元素,$\boldsymbol{C}$ 矩阵第 $i$ 行元素的值由 $k$ 次向量乘累加完成,每次计算都是先将 $\boldsymbol{A}$ 矩阵第 $i$ 行的第 $j$ 个元素扩展为值相同的向量,再与 $\boldsymbol{B}$ 矩阵的第 $j$ 行向量进行乘累加计算,每一次的向量乘累加计算是在各个 VPE 上并行进行,计算的源数据和结果数据均保存在 VPE 的本地寄存器上,每个计算结果涉及的乘累加计算均在同一个 VPE 上完成,并且 $\boldsymbol{A}$、$\boldsymbol{B}$、$\boldsymbol{C}$ 三个矩阵的数据均是按行顺序读取,访存效率高,在 $k$ 循环结束时,同时完成 $\boldsymbol{C}$ 矩阵第 $i$ 行元素值的计算.该方法能充分开发向量处理器的标量、向量协同数据加载能力,有效减少对 DDR 的存储带宽需求,能够避免低效的对乘数矩阵列向量数据的访问和各个 VPE 间的浮点归约求和计算,取得最优的内核计算性能;将处理器的一级数据缓存和阵列存储配置为 SRAM 访问模式,能够避免由于 Cache 数据不命中而导致的存储访问延迟,提高核心计算访问一级数据缓存和阵列存储的效率,采用组播 DMA 传输矩阵数据,能够显著提高从 DDR 读取矩阵数据的效率;提出依据向量处理单元 VPE 数量、VPE 的 FMAC 运算单元数量、向量存储器的容量和矩阵元素的数据类型等向量处理器体系结构特点设计最优的核心子块矩阵分块参数设计方法,能够充分开发向量处理器的多核间数据并行、核内的多 VPE 间的向量 SIMD 并行、VPE 内的多个 FMAC 单元并行、VPE 内的标、向量指令级并行等多级并行性,并根据 FMAC 指令延迟槽进行完全循环展开,让内核始终以峰值速度运行;提出基于两级 DMA 双缓冲数据搬移策略,优化和平滑多级存储结构间的数据传输,使得 DMA 的数据搬移时间完全重叠于内核的计算时间中,让整个矩阵计算以接近内核计算的速度运行,实现最优的计算性能和效率.在 MATRIX2 上的实验结果表明,提出的双精度矩阵乘法的性能达到 1106.88 GFLOPS,效率为 96.08%,核心计算的效率达到 99.39%.

**关键词** 多核向量处理器;高性能计算;矩阵乘法;分块矩阵;向量化

**中图法分类号** TP391 **DOI 号** 10.11897/SP.J.1016.2018.02251

# Vectorization of Matrix Multiplication for Multi-Core Vector Processors

LIU Zhong TIAN Xi

(*College of Computer, National University of Defense Technology, Changsha* 410073)

**Abstract** Dense matrix multiplication is one of the core computations in many algorithms from large scientific computing. An efficient vectorization of matrix multiplication for multi-core vector processors was presented. A vectorization of matrix multiplication according to row computation were presented. The basic idea of the vectorization method is that the one row elements of the $\boldsymbol{C}$ matrix is calculated at the same time. The value of the $i$-th row elements of the $\boldsymbol{C}$ matrix is completed by $k$ vector multiply and accumulate operations. For each calculation, we extend the $j$ th element of the $i$-th row of the $\boldsymbol{A}$ matrix into the vector of the same value, and then multiply and accumulate the $j$ th row elements of the $\boldsymbol{B}$ matrix. Each vector multiply and accumulate calculation is carried out in parallel on each VPE. The calculated source data and the result data are stored in the local registers of VPE, each involved multiply and accumulate operation of calculation results

收稿日期:2016-10-11;在线出版日期:2017-06-20.本课题得到国家自然科学基金(61572025,61472432)资助.刘 仲,男,1971 年生,博士,研究员,主要研究领域为高性能计算、并行计算、性能优化.E-mail:zhongliu@nudt.edu.cn.田 希,男,1986 年生,硕士研究生,主要研究方向为高性能计算、算法分析与设计.

are completed on the same VPE. The $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ matrix data are read in line order, which achieve a higher access efficiency, the calculation of the values of the $i$-th row element of the $\boldsymbol{C}$ matrix is completed at the end of the $k$ cycle. This method fully exploits scalar and vector collaborative data loading capacity of vector processor and effectively reduces the storage bandwidth requirements for DDR, it avoids low efficiency data access to column vectors of multiplier matrix and float reduction summation calculation among all VPEs, and achieves optimization kernel computation performance. The level-1 data cache and array memory of vector processor was configured as SRAM access pattern, which can avoid the storage access delay caused by the cache data miss and improve the access efficiency of core computing to the level-1 data cache and array memory, it use multicast DMA to transfer matrix data, which significantly improves the efficiency of reading matrix data from DDR. An optimized core sub-block matrix blocking method was designed based on the vector processor architecture features including the number of vector processing unit VPE, the number of FMAC operation units of VPE, the capacity of vector memory and the data type of matrix elements, which fully exploits data parallelism of multi-core vector processors, vector SIMD parallelism between multiple VPEs, parallelism of multiple FMAC elements within VPE, scalar and vector instruction level parallelism in VPEs, it make full of looping expansion in accordance with the FMAC instruction delay slot, so that the kernel computing is always running at peak speed. A data transfer strategy based on two-level DMA double buffering scheme was designed to optimize and smooth the data transfers between multilevel storage architecture, which makes kernel computation and DMA data transfer fully overlapped, so that the whole matrix calculate is always running close to the kernel computing speed and achieve optimal computing performance and efficiency. Experimental results on MATRIX2 show that the performance of presented double precision matrix multiplication achieves 1106.88 GFLOPS, an efficiency of 96.08%, and the efficiency of kernel computation achieves 99.39%.

**Keywords** multi-core vector processor; high performance computing; matrix multiplication; blocked matrix; vectorization

## 1 引 言

随着功耗和散热问题的日益突出，能耗逐步成为影响高性能计算系统重要的因素，并使得处理器的体系结构朝着多核、众核、异构 GPU、嵌入式芯片等方向发展. 最近一期的 TOP500 排行榜显示使用 6 核及以上、8 核及以上多核处理器的机器已经分别上升到 88%和 67%，有 42 台超级计算机系统采用异构 GPU 技术[1]. 为克服能耗瓶颈问题，MontBlanc① 首次将嵌入式芯片引入高性能计算领域，TI(Texas Instruments)也凭借 DSP 的低功耗优势将多核 DSP(C66x)引入高性能计算领域[2]. MATRIX2 是一款基于超长指令字(Very Long Instruction Word，VLIW)的高性能浮点多核向量处理器，它通过在单颗芯片上集成多个采用向量运算部件的向量处理器核，向量处理器核上集成丰富的乘加运算单元，使得该芯片能够在较低功耗的情况下显著提高整个芯片的峰值计算能力，非常适合加速高性能计算. 但是由于向量处理器的体系结构复杂，存储层次多，涉及指令级并行、向量计算、数据级并行、多核并行等多个层次的并行化，简单的移植现有的面向单核处理器设计的应用程序和算法难以充分发挥向量处理器的峰值计算性能[3].

BLAS(Basic Linear Algebra Subroutines)算法库是大规模科学计算最常用的核心数学算法库之一，工业界针对各自的处理器平台都推出了高度优化的 BLAS 实现，如 IBM 的 ESSL、Intel 的 MKL、AMD 的 ACML 等. 针对不同处理器的体系结构特点实现 BLAS 库的优化设计一直是国内外学术界的研究热点，例如，高性能的 GotoBLAS② 库[4] 是针对不同体

① http://www.montblance-project.eu/

② GotoBLAS homepage. http://www.tacc.utexas.edu/tacc-projects/gotoblas2

系结构特点采用手工汇编高度优化实现的，ATLAS① 库是采用自适应优化技术设计的，张先轶等针对多款处理器开发了高性能的 OpenBLAS 库[5]等. Sohl 等人[6]面向一种异构并行 DSP 体系结构研究了大规模矩阵乘法，研究表明针对该款处理器新颖的并行结构和存储系统优化设计的矩阵乘法映射能够有效的隐藏数据访问开销. 张凯面向向量 SIMD DSP 研究了高效矩阵运算技术[7]，主要研究高效能矩阵计算对 DSP 体系结构的设计需求、如何基于软硬件协同优化技术支持稀疏矩阵计算、对嵌入式领域高性能矩阵计算提供支持的矩阵寄存器文件设计等.

BLAS 库的最常用核心算法是矩阵乘法(General Matrix-Matrix Multiplication，GEMM)，GEMM 算法是非常典型的计算密集型和访存密集型算法应用(由于稀疏矩阵有特殊的考虑，本文讨论的矩阵乘法都是指稠密矩阵乘法)，在高性能基准测试程序(High Performance Linpack，HPL)的运算时间占比超过 90%[8]，其计算效率能够较好的评测处理器的实际运算能力. 因此，针对处理器的体系结构特点研究 GEMM 优化方法对评测该处理器的计算效率、发挥处理器的计算优势和提高应用程序的运行速度均具有很重要的参考价值.

如图 1 所示，HPL 常采用分块 LU 分解方法以求解稠密线性方程组. 以 $n$ 阶方阵为例，假定基本块大小为 NB. 整个 LU 分解分成 4 个步骤：(1) 基本块分解为：$L_{i-1}U_{i-1}$；(2) 行矩阵更新：$U_i=L_{i-1}^{-1}\times U_i$；(3) 列矩阵更新：$L_i=L_i\times U_{i-1}^{-1}$；(4) 尾矩阵更新：$A_i=A_i-L_i\times U_i$.

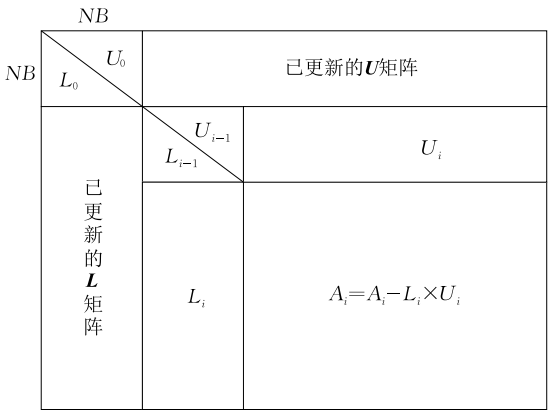



图 1　块 LU 分解

依次循环上述 4 个过程，直至完成整个矩阵的 LU 分解. 其中尾矩阵的更新大约占总运算量的 90%，而尾矩阵的更新运算就是本文讨论的 GEMM 算法，所以 GEMM 是否能够高效的实现对 HPL 的计算效率有着重要的影响.

分块矩阵乘法方法是提升大规模 GEMM 性能的主要方法，最典型的研究是 Gunnels 针对基于 Cache 的多级存储结构，提出了分层计算的 GEMM 优化方法[9]，能够降低多级存储层次间搬运数据的平均开销，Volkov 等人研究了 GPU 平台的 BLAS 优化实现[10]，Igual 等人针对 TI 的 C66x 多核 DSP 研究 GEMM 对 BLAS-3 的优化实现[11].

这类分块矩阵乘法的主要方法是将大矩阵的乘法分割为一系列子矩阵的乘法，设 $M$ 为 Cache 的容量，则分块参数中的子矩阵块大小 *blocksize* 通常满足约束条件 $blocksize<=\mathrm{sqrt}(M/3)$，若子矩阵计算时的数据访问能够全部在 Cache 中命中，则可以使得该部分的计算能以接近峰值的性能进行，从而提高整个大矩阵乘法的计算性能. 然而，这类方法中，由于多级 Cache 均可能存在数据不命中情况，计算过程中易产生 Cache 数据不命中，Cache 容量的限制等问题使得 Cache 数据不命中情况大量存在，由此产生的存储访问延迟将显著降低 GEMM 的计算性能和效率.

本文针对多核向量处理器的体系结构特点，提出一种基于 SRAM 的分块矩阵乘法算法优化方法，将 L1D 配置为 SRAM 模式，通过 DMA 在后台操作 DDR 和 SRAM 之间的数据搬移，采用两级 DMA 双缓冲机制以及优化的分块参数设计使得数据搬移时间隐藏于计算时间中，完全避开了 Cache 数据不命中而导致的存储访问延迟；同时依据向量处理单元 VPE 的数量、每个 VPE 的 FMAC 运算单元的数量、向量存储器的容量和矩阵元素的数据类型等向量处理器体系结构特点提出一种高度优化的核心子块矩阵参数设计方法，并提出一种按行计算的矩阵乘法向量化方法实现核心子块矩阵的计算，能够避免对矩阵列向量数据的访问和向量处理单元之间的浮点归约求和计算，取得非常接近处理器峰值的计算性能. 实验测试结果表明，本文提出的方法能够取得比传统的基于 Cache 的方法更高的计算性能和效率.

## 2　MATRIX2 的体系结构

MATRIX2 是国防科大自主研发的一款面向高密度计算的高性能浮点多核向量处理器. 该处理器集成了 12 颗向量处理器内核，在主频 1GHz 下双精

① ATLAS homepage. http://math-atlas.sourceforge.net/

度峰值性能为 1152 GFLOPS.

MATRIX2 的单核体系结构如图 2 所示，向量处理器内核由标量处理部件(Scalar Processing Unit，SPU)和向量处理部件(Vector Processing Unit，VPU)组成，其中，SPU 负责标量计算和流控，SPU 和 VPU 之间设计了交换数据的共享寄存器，SPU 提供广播指令能够将标量寄存器的数据广播到 VPU 的向量寄存器. 向量处理器核支持 11 发射且长度可变的 VLIW 指令，指令执行包最多支持 5 条标量指令和 6 条向量指令，由指令派发单元对指令执行包进行识别并派发到相应的功能单元去执行. VPU 提供主要的向量计算能力，其集成了 16 个向量处理单元(Vector Processing Element，VPE)，每个 VPE 包括一个含 64 个 64 位寄存器的局部寄存器文件，同时，VPE 的所有相同编号的局部寄存器在逻辑上构成一个 1024 位的向量寄存器. 每个 VPE 包括 3 个 FMAC 运算单元、1 个 BP 单元和 2 个 L/S 数据访问单元. 派发到各 VPE 上的向量指令同时独立执行，每个时钟周期同时支持 3 个双精度浮点乘加运算并发执行. 2 个向量 L/S 数据访问单元同时支持 2 路 2048 位向量数据的 Load/Store. MATRIX2 的标量存储器包括一级程序缓存 L1P 和一级数据缓存 L1D，其中 L1D 可配置为全 Cache、全 SRAM 和两者混合三种访存模式；MATRIX2 提供 768 k 的大容量阵列存储器(Array Memory，AM)用于向量数据访问，并且提供丰富的直接存储访问(Direct Memory Access，DMA)模式实现各级存储空间的快速访问. MATRIX2 的 12 个核之间通过全局高速缓存(Global Cache，GC)和共享 DDR 存储器实现核间的数据共享和交换.

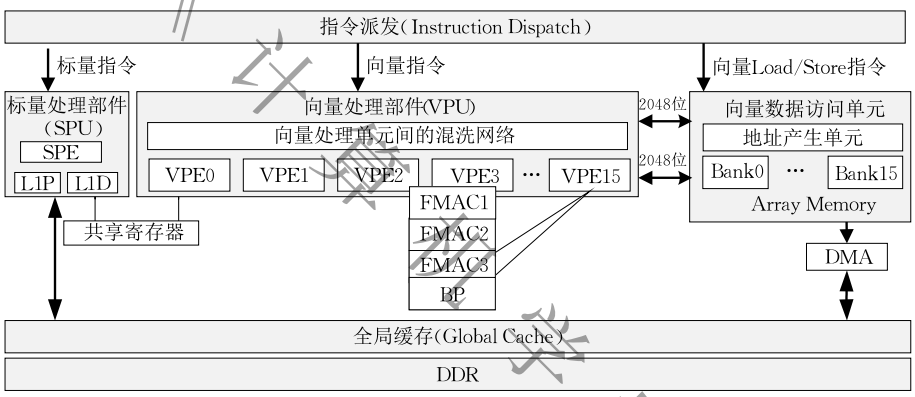



图 2 MATRIX2 单核体系结构

## 3 多核并行矩阵乘法算法设计

多核 MATRIX2 采用共享 GC 和 DDR 的存储结构，分块矩阵乘法的不同子块间的计算不需要紧耦合的数据交换，适合数据并行. 如图 3 所示，基于分块矩阵乘法的基本原理，把矩阵 $\boldsymbol{B}$、$\boldsymbol{C}$ 按列从逻辑上划分为 $P$ 块，每个核计算的列数为 $n\_core=n/P$(假定 $n$ 是 $P$ 的倍数).

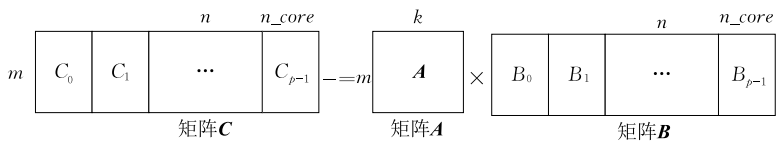
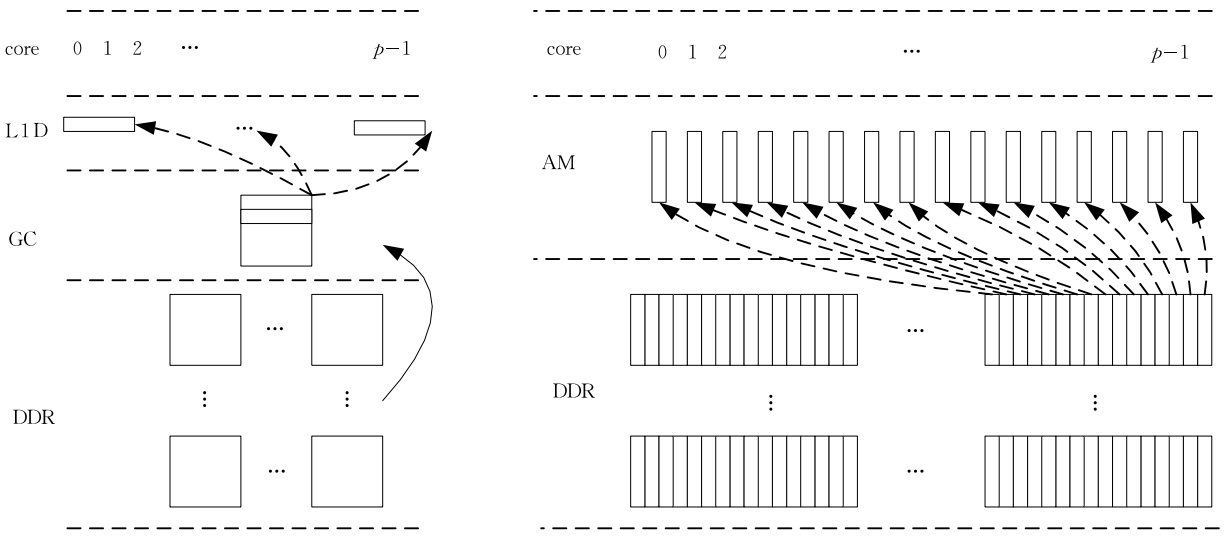



图 3 多核并行分块矩阵计算方法

$\boldsymbol{A}$ 矩阵分块方法如图 4(a)所示，$\boldsymbol{A}$ 矩阵在 DDR 中按照设定的分块大小连续存储，每次将其中的一个分块从 DDR 载入全局缓冲区 GC 中，由所有核共享，各核基于数据并行的方法分别加载 $\boldsymbol{B}$、$\boldsymbol{C}$ 的子矩阵数据，并行计算 $\boldsymbol{C}_i=\boldsymbol{C}_i-\boldsymbol{A}\times\boldsymbol{B}_i$. 每个核独立的从 GC 中读取更小的分块到自己的 L1D 中参与 $\boldsymbol{B}$、$\boldsymbol{C}$ 子块的计算，直到所有核对 GC 中 $\boldsymbol{A}$ 矩阵子块的计算完毕，再载入新的 $\boldsymbol{A}$ 矩阵子块. 各个核对 L1D 中的 $\boldsymbol{A}$ 矩阵子块数据读取时是通过标量存储加载，再通过标量的广播指令传输到向量寄存器中参与 $\boldsymbol{B}$、$\boldsymbol{C}$ 矩阵数据的乘加计算，这种方法能够显著降低对 DDR 的数据带宽需求. $\boldsymbol{B}$、$\boldsymbol{C}$ 矩阵采用如图 4(b)所示的矩阵分块方法，在 DDR 中将 $\boldsymbol{B}$、$\boldsymbol{C}$ 矩阵按照设定的分块大小连续存储，每次读取 DDR 中连续的




(a) ***A***矩阵分块方法　　(b) ***B***、***C***矩阵分块方法

图 4　矩阵分块方法

一个 ***B***、***C*** 矩阵分块数据，该分块包含若干个更小的小子块，通过组播 DMA 将各个不同的小子块分发到不同核的向量存储 AM 中. 这种分块方法能够显著提高 DDR 读取矩阵数据的效率，有效保证数据传输时间隐藏于计算时间中.

### 3.1　多核并行矩阵乘法算法

结合第 4 节单核上的矩阵乘法算法，多核并行矩阵乘法算法的框架如算法 1 所示. 其中包含两级基于 DMA 双缓冲的数据搬移优化方法：(1) 在 $n$ 维方向基于 DMA 双缓冲机制计算 ***A*** 的子块 $m\_gc\times k\_gc$ 与 ***B***、***C*** 的子块 $k\_gc\times n\_core$ 的 GEMM 计算；(2) 在 $m$ 维方向基于 DMA 双缓冲机制计算 ***A*** 的子块 $m\_L1D\times k\_gc$ 与 ***B***、***C*** 的子块 $k\_gc\times n\_am$ 的 GEMM 计算. 以 $n$ 维方向的基于 DMA 双缓冲机制的 GEMM 算法为例如算法 2 所示.

**算法 1.**　多核并行矩阵乘法算法.

输入：$\boldsymbol{A}_{m\times k}$，$\boldsymbol{B}_{k\times n}$，$\boldsymbol{C}_{m\times n}$

输出：$\boldsymbol{C}_{m\times n}$

BEGIN

根据 AM 的容量设立矩阵 ***B***、***C*** 的子块大小 $k\_gc\times n\_am$，$n$ 维按列划分为 $P\times p$ 块，每个核计算的块数为 $p$，对所有处理器核(0，…，$P-1$)同时执行如下的算法：

根据 GC 的容量设立矩阵 ***A*** 的子块大小 $m\_gc\times k\_gc$，矩阵 ***A*** 在 $m$ 维和 $k$ 维划分的子块数分别为 $r$，$q$

FOR $i=0$ TO $r$ DO

FOR $j=0$ TO $q$ DO

搬移矩阵 ***A*** 的子块 $A_{ij}$ 至 GC；

所有处理器核(0，…，$P-1$)同时独立执行如下的算法：

在 $n$ 维方向基于 DMA 双缓冲机制计算 ***A*** 的子块 $m\_gc\times k\_gc$ 与 ***B***、***C*** 的子块 $k\_gc\times n\_core$ 的 GEMM 计算

{

所有处理器核(0，…，$P-1$)独立执行如下的算法：

对 ***A*** 子块 $m\_gc\times k\_gc$ 在 $m$ 维进行划分，子块大小为 $m\_L1D\times k\_gc$；

在 $m$ 维方向基于 DMA 双缓冲机制计算 ***A*** 的子块 $m\_L1D\times k\_gc$ 与 ***B***、***C*** 的子块 $k\_gc\times n\_am$ 的 GEMM 计算；

}

所有处理器核在此执行栅栏同步，直到所有核完成 ***A*** 子块 $m\_gc\times k\_gc$ 的计算；

END FOR

END FOR

END

**算法 2.**　$n$ 维方向的基于 DMA 双缓冲机制的 GEMM 算法.

输入：$\boldsymbol{A}_{m\_gc\times k\_gc}$，$\boldsymbol{B}_{k\_gc\times n\_core}$，$\boldsymbol{C}_{m\_gc\times n\_core}$

输出：$\boldsymbol{C}_{m\_gc\times n\_core}$

BEGIN

令 $q=n\_core/n\_am$，在 AM 中为 ***B*** 矩阵子块设置容量为 $k\_gc\times n\_am$ 的两个缓冲区 Bbuffer0 和 Bbuffer1，在 AM 中为 ***C*** 矩阵子块设置容量为 $m\_gc\times n\_am$ 的两个缓冲区 Cbuffer0 和 Cbuffer1，设置通过 DMA 从 DDR 传输 ***B***、***C*** 矩阵子块数据到缓冲区 Bbuffer0、Cbuffer0，设置通过 DMA 从 DDR 传输 ***B***、***C*** 矩阵子块数据到缓冲区 Bbuffer1、Cbuffer1.

FOR ($i=0$；$i<q-2$；$i+=2$；) {

等待缓冲区 Bbuffer0、Cbuffer0 数据就位；

在缓冲区 Bbuffer0、Cbuffer0 执行 ***A*** 子块 $m\_gc\times k\_gc$ 与 ***B*** 子块 $k\_gc\times n\_am$、***C*** 子块 $m\_gc\times n\_am$ 的

GEMM 计算；
设置通过 DMA 传输 $\boldsymbol{C}$ 矩阵子块在 Cbuffer0 的结果数据到 DDR；
设置通过 DMA 从 DDR 传输 $\boldsymbol{B}$、$\boldsymbol{C}$ 矩阵子块数据到缓冲区 Bbuffer0、Cbuffer0；
等待缓冲区 Bbuffer1、Cbuffer1 数据就位；
在缓冲区 Bbuffer1、Cbuffer1 执行 $\boldsymbol{A}$ 子块 $m\_gc\times k\_gc$ 与 $\boldsymbol{B}$ 子块 $k\_gc\times n\_am$、$\boldsymbol{C}$ 子块 $m\_gc\times n\_am$ 的 GEMM 计算；
设置通过 DMA 传输 $\boldsymbol{C}$ 矩阵子块在 Cbuffer1 的结果数据到 DDR；
设置通过 DMA 从 DDR 传输 $\boldsymbol{B}$、$\boldsymbol{C}$ 矩阵子块数据到缓冲区 Bbuffer1、Cbuffer1；
}
END FOR
等待缓冲区 Bbuffer0、Cbuffer0 数据就位；
在缓冲区 Bbuffer0、Cbuffer0 执行 $\boldsymbol{A}$ 子块 $m\_gc\times k\_gc$ 与 $\boldsymbol{B}$ 子块 $k\_gc\times n\_am$、$\boldsymbol{C}$ 子块 $m\_gc\times n\_am$ 的 GEMM 计算；
设置通过 DMA 传输 $\boldsymbol{C}$ 矩阵子块在 Cbuffer0 的结果数据到 DDR；
等待缓冲区 Bbuffer1、Cbuffer1 数据就位；
在缓冲区 Bbuffer1、Cbuffer1 执行 $\boldsymbol{A}$ 子块 $m\_gc\times k\_gc$ 与 $\boldsymbol{B}$ 子块 $k\_gc\times n\_am$、$\boldsymbol{C}$ 子块 $m\_gc\times n\_am$ 的 GEMM 计算；
设置通过 DMA 传输 $\boldsymbol{C}$ 矩阵子块在 Cbuffer1 的结果数据到 DDR；
END

### 3.2 GEMM 性能模型

根据 GC 的容量设置矩阵 $\boldsymbol{A}$ 在 GC 中的子块 $m\_gc\times k\_gc$，根据 AM 的容量设置矩阵 $\boldsymbol{B}$、$\boldsymbol{C}$ 在 AM 中的子块 $k\_gc\times n\_am$. 计算 $\boldsymbol{A}$ 子块 $m\_gc\times k\_gc$ 与 $\boldsymbol{B}$、$\boldsymbol{C}$ 子块 $k\_gc\times n\_am$ 的计算开销 $t$.

设 $t0$ 为最内层 $\boldsymbol{A}$ 的子块 $m\_L1D\times k\_gc$ 与 $\boldsymbol{B}$、$\boldsymbol{C}$ 子块 $k\_gc\times n\_am$ 的计算时间，4.5 节分析了该部分计算是以接近处理器的峰值计算的；$t1$ 为 $\boldsymbol{A}$ 子块 $m\_gc\times k\_gc$ 填充 GC 时间；$t2$ 为第一次 $\boldsymbol{B}$、$\boldsymbol{C}$ 子块 $k\_gc\times n\_am$ 通过 DMA 搬入 AM 的时间；$t3$ 为最后一次结果 $\boldsymbol{C}$ 子块 $k\_gc\times n\_am$ 通过 DMA 搬出时间；$t4$ 为 DMA 开启和 POLL 时间；$t5$ 为多核同步时间；$u=m\_gc/m\_L1D$；依据 4.3 节提出的基于两级 DMA 双缓冲的数据搬移优化策略，则计算 $\boldsymbol{A}$、$\boldsymbol{B}$、$\boldsymbol{C}$ 一个子块的计算时间开销为

$$t=t1+t2+t3+(t0+t4)\times u+t5.$$

进一步考虑更大规模的矩阵乘法，如图 5 所示，采用分块矩阵乘法，令 $m=m\_gc\times r$，$k=k\_gc\times p$，$n=n\_am\times q\times P$，则总计算时间为 $t\times p\times q\times r$，总的计算量为 $2\times m\times k\times n$，因此，根据测试时间可得到的 GEMM 计算性能为 $2\times m\times k\times n/(t\times p\times q\times r)$，计算效率为计算性能除以处理器的峰值性能.

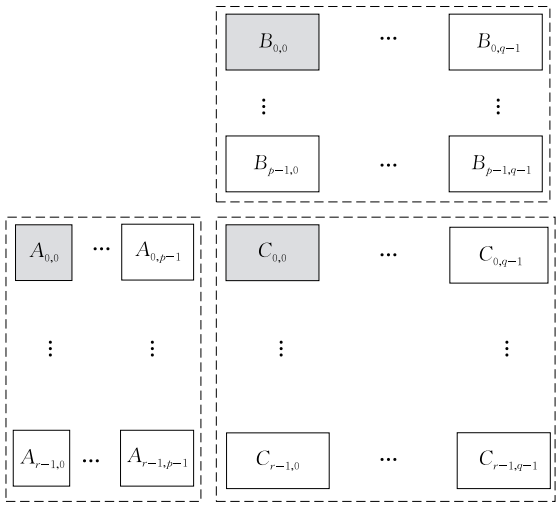



图 5 分块矩阵乘法

### 3.3 本文创新点

与传统的分块矩阵乘法计算相比，本文提出的矩阵乘法向量化方法能够大幅度提高多核向量处理器上的矩阵乘法计算性能和效率，具有以下几个显著优点：

(1) 核心计算访问的 L1D 和 AM 均采用 SRAM 模式，完全避开了 Cache 数据不命中而导致的存储访问延迟；

(2) $\boldsymbol{A}$ 矩阵虽然通过 GC 共享，但对程序员透明，通过 DMA 在后台操作 DDR 和 SRAM 之间的数据搬移，采用两级 DMA 双缓冲机制、优化的分块参数设计以及组播 DMA 传输方法，能够显著提高 DDR 读取数据效率，使得数据搬移时间隐藏于计算时间中；

(3) 依据向量处理单元 VPE 的数量、每个 VPE 的 FMAC 运算单元的数量、向量存储器的容量和矩阵元素的数据类型等向量处理器体系结构特点设计最优的核心子块矩阵参数，能够充分开发向量处理器的多级并行性，包括多核间的数据并行、核内的多 VPE 间的 SIMD 并行、单 VPE 内的 $c$ 个 FMAC 单元完全并行、单 VPE 内的每个 FMAC 单元根据 FMAC 指令延迟槽进行完全循环展开、单 VPE 内的标、向量指令级并行等，让内核始终以峰值速度运行；

(4) 基于按行计算的矩阵乘法向量化方法实现核心子块矩阵的计算，能够避免通常向量处理器不支持的对乘数矩阵列向量数据的访问，以及硬件开销较大的向量处理单元之间的浮点求和归约设计，

取得非常接近处理器峰值的计算性能.

## 4　单核上的矩阵乘法算法

Goto 的研究表明分块参数的选取及核心计算的设计与体系结构密切相关[6]. 为评估和优化性能，我们通过分块参数建立分块矩阵乘法的性能模型，在满足体系结构的硬件约束条件下，求解最优化的分块参数，使得分块矩阵乘法的性能最大化. 为方便描述，设定参数如下：

(1) 体系结构参数. 设多核处理器的核数为 $P$，每个核上 VPE 的数量为 $v$，L1D、AM、GC 的存储容量分别为 $S_1$、$S_2$、$S_3$，每个 VPE 上 FMAC 单元的数量为 $c$.

(2) 分块矩阵参数. 设矩阵乘法计算：$\boldsymbol{C}=\boldsymbol{C}-\boldsymbol{A}\times\boldsymbol{B}$，其中矩阵 $\boldsymbol{A}$、$\boldsymbol{B}$、$\boldsymbol{C}$ 规模分别是 $m\times k$，$k\times n$，$m\times n$，矩阵元素的数据大小 $d$.

### 4.1　单核上的矩阵分块方法

在多核并行 GEMM 计算中，矩阵 $\boldsymbol{A}$ 是每个核都需要访问的，若将矩阵 $\boldsymbol{A}$ 驻留在共享 GC 中，则矩阵乘法计算只需要搬移矩阵 $\boldsymbol{B}$、$\boldsymbol{C}$ 的数据，将显著减轻对 DDR 的带宽需求，因此需要将矩阵 $\boldsymbol{A}$ 一直驻留在 GC 中，直到与之相关的计算全部完成. 另一方面，由于 GC 的存储容量有限，通常远小于矩阵 $\boldsymbol{A}$ 的数据量，所以需要进一步分块，使得分块计算的矩阵 $\boldsymbol{A}$ 的子块能够完全在 GC 中命中. Goto 的研究表明[6]，在相当矩阵规模下，基于 GEPP (General Panel-Panel Multiplication) 实现 GEMM，基于 GEBP (General Block-Panel Multiplication) 实现 GEPP 是效率较高的一种分块方式，非常适合 $m$ 和 $n$ 都非常大且 $k$ 不是很小的情况，这也是 HPL 中矩阵更新时的典型参数特征.

在单核计算 $\boldsymbol{C}_i=\boldsymbol{C}_i-\boldsymbol{A}\times\boldsymbol{B}_i$ 时，采用如图 6 所示的单核分块 GEMM 方法，其中外层循环是对矩阵 $\boldsymbol{A}$ 的 $k$ 维方向进行划分，采用 GEPP 的分块方式计算 GEMM，内层循环是对矩阵 $\boldsymbol{A}$ 的 $m$ 维方向进行划分，采用 GEBP 的分块方式计算 GEPP. 子块 $m\_gc\times k\_gc$ 是矩阵 $\boldsymbol{A}$ 能够存放在 GC 中的最大子块，该子块一直驻留在 GC 中，直到矩阵 $\boldsymbol{B}$、$\boldsymbol{C}$ 的子块 $k\_gc\times n\_core$ 的计算全部完成.

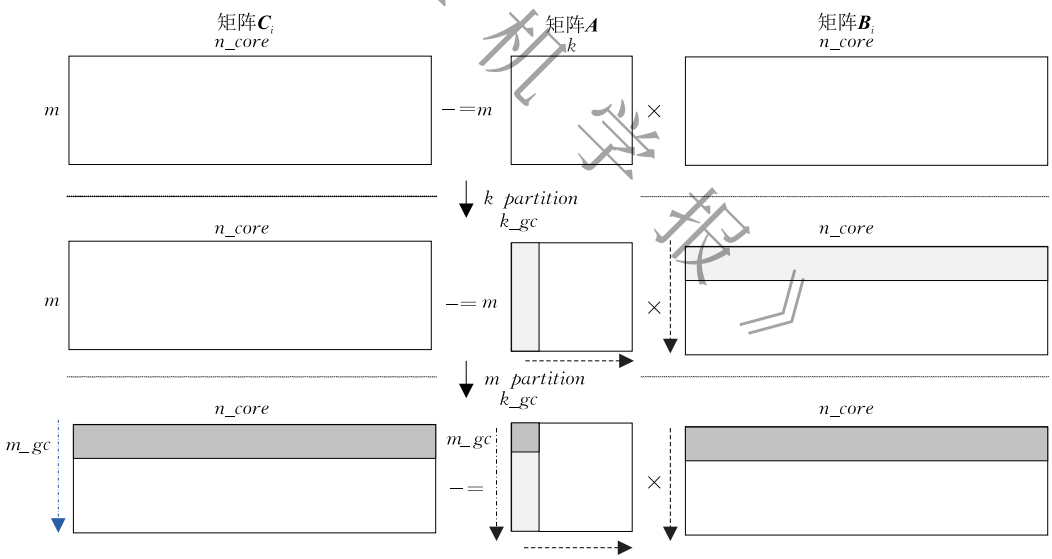



图 6　单核分块矩阵乘法方法

### 4.2　kernel 级的矩阵分块方法

在上述单核分块矩阵乘法方法的 GEBP 计算中，通常矩阵 $\boldsymbol{B}$、$\boldsymbol{C}$ 的子块 $k\_gc\times n\_core$ 的数据量远大于 AM 的存储容量，需要进一步分块，使得更小粒度的分块能够存储在 AM 中，确保 AM 中的分块矩阵计算能够以处理器的峰值性能进行计算.

单核的 GEBP 计算采用如图 7 所示 kernel 级分块矩阵计算方法. 其中外层循环是对矩阵 $\boldsymbol{B}$、$\boldsymbol{C}$ 的 $n$ 维方向进行划分，使得矩阵 $\boldsymbol{B}$、$\boldsymbol{C}$ 的子块 $k\_gc\times n\_am$ 能够存放在 AM 存储中；内层循环是对矩阵 $\boldsymbol{A}$ 的 $m$ 维方向进行划分，使得 $\boldsymbol{A}$ 矩阵的子块 $m\_L1D\times k\_gc$ 能够存放在配置为 SRAM 模式的 L1D 中. 矩阵 $\boldsymbol{A}$ 的子块 $m\_L1D\times k\_gc$ 与矩阵 $\boldsymbol{B}$、$\boldsymbol{C}$ 的子块 $k\_gc\times n\_am$ 是最小的 kernel 分块矩阵计算，所需数据直接以 Load/Store 指令加载到寄存器文件，通过软件流水优化后，不会有存储访问延迟，使得该部分计算完全以处理器的峰值速度进行计算.

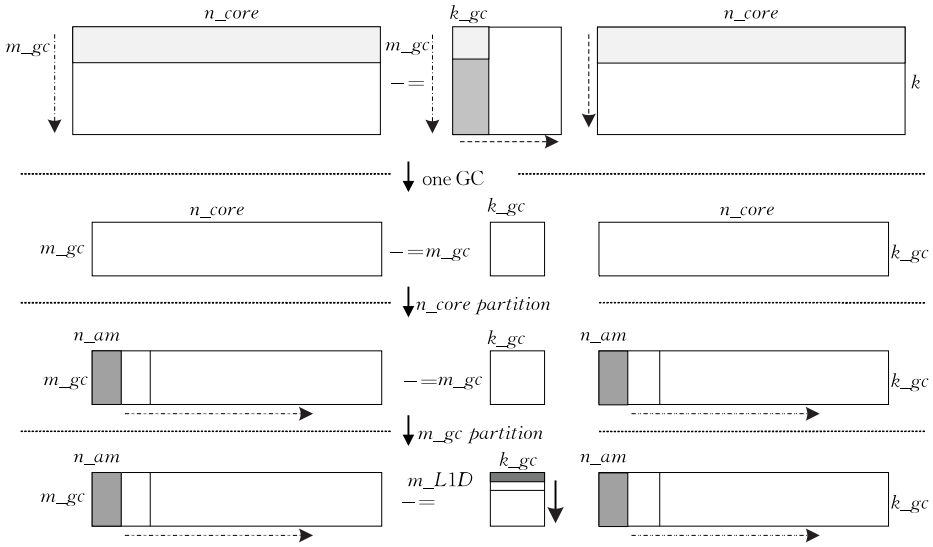



图 7 kernel 级分块矩阵乘法方法

### 4.3 基于两级 DMA 双缓冲的数据搬移优化策略

平滑各级存储之间的数据搬移开销是提高 GEMM 计算效率的关键因素之一. MATRIX2 的数据存储涉及寄存器文件、L1D 和 AM、GC、DDR 多级存储结构,我们提出基于两级 DMA 双缓冲的数据搬移优化策略,将 L1D 配置为 SRAM 模式,在 SRAM 中为矩阵 ***A*** 的子块设立两个缓冲区,在 *AM* 中为矩阵 ***B***、***C*** 的子块设立两个缓冲区,采用 DMA 双缓冲的乒乓方式实现核心计算与 DMA 数据搬移重叠,隐藏数据搬移时间,从而提高 GEMM 的计算效率. 具体数据搬移策略如图 8 所示.

(1) 在第 1 栏中,核心计算在启动前首先准备好第一个缓冲区的数据. 假定矩阵 ***A*** 的子块已经驻留在 GC 中,由 DMA 搬移矩阵 ***B***、***C*** 的子块数据至 AM 的第一个缓冲区,搬移矩阵 ***A*** 的子块数据至 SRAM 中的第一个缓冲区;

(2) 在第 2 栏中,核心首先读取 AM 和 SRAM 中已经准备好的第一个缓冲区数据进行计算,同时启动 DMA 搬移矩阵 ***B***、***C*** 的子块数据至 AM 的第二个缓冲区,搬移矩阵 ***A*** 的子块数据至 SRAM 中的第二个缓冲区. 其中后者优先级高,在 AM 的第一个缓冲区数据计算过程中,SRAM 将对矩阵 ***A*** 在 GC 中的全部子块完成一次遍历,在此过程中,SRAM 的两个缓冲区的计算与 DMA 数据搬移完全重叠,结束时完成 AM 的第一个缓冲区的数据计算;

(3) 在第 3 栏中,核心首先读取 AM 中已经准备好的第二个缓冲区数据进行计算,同时由 DMA 将 AM 的第一个缓冲区的计算结果搬移至 DDR,期间,SRAM 的两个缓冲区的计算与 DMA 数据搬移完全重叠;

(4) 在第 4 栏中,核心继续读取 AM 中已经准备好的第二个缓冲区数据进行计算,同时启动 DMA 搬移矩阵 ***B***、***C*** 的子块数据至 AM 的第一个缓冲区. 在 AM 的第二个缓冲区数据的计算过程中,SRAM 将重新对矩阵 ***A*** 在 GC 中的全部子块完成一次遍历,SRAM 的两个缓冲区的计算与数据搬移完全重叠,结束时完成 AM 的第二个缓冲区数据计算.

上述过程一直循环进行下去,直到矩阵 ***B***、***C*** 在 $n$ 维方向的全部子块计算完毕. 矩阵 ***A*** 在 GC 中的

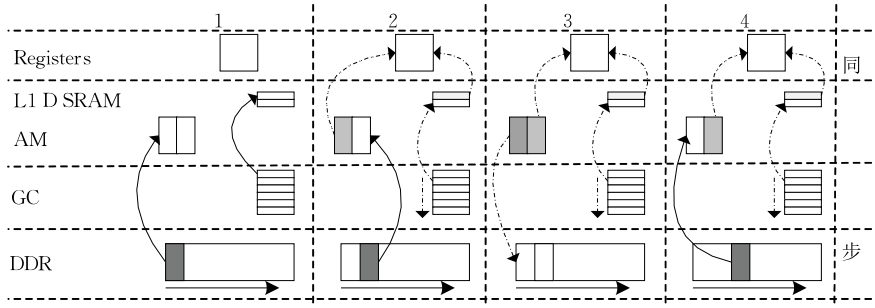



图 8 GEBP 分块中数据的搬移策略

子块完成计算后，所有核需要同步一次，确保每个核完成该次的 GEBP 计算，并启动 DMA 搬移矩阵 $\boldsymbol{A}$ 的新子块到 GC 中，开启新的 GEBP 计算，直到完成全部计算.

### 4.4　按行计算的核心 GEMM 向量化方法

矩阵乘法 $\boldsymbol{C}=\boldsymbol{C}-\boldsymbol{A}\times\boldsymbol{B}$ 通常采用式(1)计算：

$$C_{ij}=C_{ij}-\sum_{l=0}^{k-1}A_{il}\times B_{lj},$$
$$i=0,1,\cdots,m-1;\ j=0,1,\cdots,n-1 \tag{1}$$

在传统的三重循环实现中，$\boldsymbol{C}$ 矩阵的元素 $C_{ij}$ 都是由 $\boldsymbol{A}$ 矩阵的第 $i$ 行元素与 $\boldsymbol{B}$ 矩阵的第 $j$ 列元素进行点积计算而得，尽管这种计算方法直观简单、易实现，但是在向量处理器上具有两个明显的缺点：

(1) 向量处理器通常不支持按列模式访问矩阵数据，需要提前将 $\boldsymbol{B}$ 矩阵进行转置处理；

(2) 需要对各个向量处理单元计算结果进行归约求和才能得到 $\boldsymbol{C}$ 矩阵元素的最终结果，开销较大.

我们将式(1)进行适当的变换，设 $\boldsymbol{B}$、$\boldsymbol{C}$ 矩阵分别按行分块，行向量分别用 $\boldsymbol{b}_j$，$\boldsymbol{c}_i$ 表示，则式(1)可变换为式(2)表示：

$$\begin{cases}\boldsymbol{c}_0 -\!= a_{00}\,\boldsymbol{b}_0+a_{01}\boldsymbol{b}_1+\cdots+a_{0k-1}\,\boldsymbol{b}_{k-1}\\ \boldsymbol{c}_1 -\!= a_{10}\,\boldsymbol{b}_0+a_{11}\boldsymbol{b}_1+\cdots+a_{1k-1}\,\boldsymbol{b}_{k-1}\\ \quad\vdots\\ \boldsymbol{c}_{m-1} -\!= a_{m-10}\,\boldsymbol{b}_0+a_{m-11}\,\boldsymbol{b}_1+\cdots+a_{m-1k-1}\,\boldsymbol{b}_{k-1}\end{cases} \tag{2}$$

根据式(2)，我们提出一种按行计算的 GEMM 向量化方法. 如图 9 所示，区别于传统方法每次计算 $\boldsymbol{C}$ 矩阵的一个元素，该向量化方法的基本思想是每次同时计算 $\boldsymbol{C}$ 矩阵的一行元素，计算 $\boldsymbol{C}$ 矩阵第 $i$ 行元素的值由 $k$ 次向量乘累加完成，每次计算都是先将 $\boldsymbol{A}$ 矩阵第 $i$ 行的第 $j$ 个元素扩展为值相同的向量，再与 $\boldsymbol{B}$ 矩阵的第 $j$ 行向量进行乘累加计算，每一次的向量乘累加计算是在各个 VPE 上并行进行，计算的源数据和结果数据均保存在 VPE 的本地寄存器上，每个计算结果涉及的乘累加计算均在同一个 VPE 上完成，并且 $\boldsymbol{A}$、$\boldsymbol{B}$、$\boldsymbol{C}$ 三个矩阵的数据均是按行顺序读取，访存效率高，在 $k$ 循环结束时，同时完成 $\boldsymbol{C}$ 矩阵第 $i$ 行元素值的计算. 相比传统的计算方法，该向量化方法完全避免了对 $\boldsymbol{B}$ 矩阵的列向量数据访问或者转置处理以及各个 VPE 间的归约求和计算，并且按行读取数据的存储效率更高.

```
{实现C=C-A×B}
for i=0 to m-1
    VPU read 向量c_i
    for j=0 to k-1
        SPU read 标量a_ij
        SPU将标量a_ij扩展为值相同的向量a_j
        VPU read 向量b_j
        VPU执行向量乘累加计算c_i-=a_j×b_j
    end
    VPU write back 向量c_i
end
```

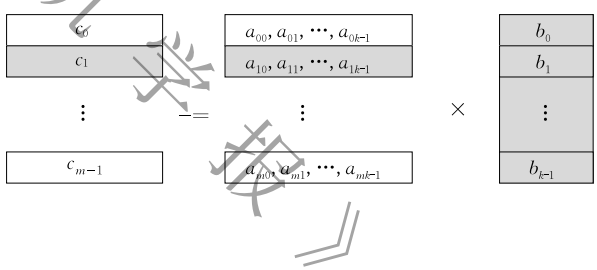



图 9　按行计算的核心 GEMM 向量化方法

基于上述按行计算的 GEMM 向量化方法，只能是 $\boldsymbol{A}$ 矩阵的数据通过标量存储读取数据，并广播到向量寄存器参与 GEMM 计算，因此设计中采用共享 $\boldsymbol{A}$ 矩阵，而非共享 $\boldsymbol{B}$ 矩阵的方法.

### 4.5　核心 GEPB 的优化设计

矩阵 $\boldsymbol{A}$ 在 GC 中的子块 $m\_gc\times k\_gc$ 与矩阵 $\boldsymbol{B}$、$\boldsymbol{C}$ 在 AM 中的子块 $k\_gc\times n\_am$ 根据 SRAM 的容量继续采用基于 GEPB(General Panel-Block Multiplication)的分块方式完成. GEPB 计算矩阵 $\boldsymbol{A}$ 的子块 $m\_L1D\times k\_gc$ 与矩阵 $\boldsymbol{B}$、$\boldsymbol{C}$ 的子块 $k\_gc\times n\_am$，是最小的 kernel 分块矩阵计算，该部分的计算是以处理器的峰值性能计算的，其性能高低直接影响大矩阵 GEMM 的性能. 首先，考虑 MATRIX2 中每个向量处理器核 VPE 的数量为 $v$，每个 VPE 的 FMAC 单元数为 $c$，基于按行计算的 GEMM 向量化方法，则 $n\_am$ 满足条件：$n\_am=i\times v(i=1,2,\cdots,c)$；其次，考虑 SRAM、AM 和 GC 的容量限制，则在矩阵 $\boldsymbol{A}$、$\boldsymbol{B}$ 和 $\boldsymbol{C}$ 子块参数必须满足如下约束条件：

$$\begin{cases}2\times m\_L1D\times k\_gc\times d\le S_1\\ 2\times k\_gc\times n\_am\times d\le S_2\\ 2\times m\_gc\times k\_gc\times d\le S_3\end{cases} \tag{3}$$

从我们实际的软件流水优化性能看，为减少循环开销，让 VPE 的 $v$ 个 FMAC 指令同时并行执行，并根据 FMAC 指令延迟槽进行循环展开，能够取得最佳的程序性能. 我们提出一种最优的分块参数设计方法如下：

(1) 与传统的根据 Cache 的容量确定子块矩阵

参数的方法不同,而是根据向量处理器的向量处理单元 VPE 的数量 $v$ 和每个 VPE 中的 FMAC 运算单元的数量 $c$,向量存储器的容量 $S_2$ 和矩阵元素的数据大小 $d$,确定最优的子块矩阵参数;

(2)子块矩阵参数按照如下方法确定,乘数矩阵 $\boldsymbol{B}$ 和结果矩阵 $\boldsymbol{C}$ 的子块的列数为 $n\_am=v\times c$,行数 $k\_gc$ 根据约束条件(3)取最大整数值;

(3)被乘数矩阵 $\boldsymbol{A}$ 的子块矩阵的行数等于乘数矩阵 $\boldsymbol{B}$ 的子块矩阵的行数,即 $m\_gc$ 根据约束条件(3)取最大整数值;

(4)矩阵 $\boldsymbol{A}$ 在 SRAM 中的子块行数 $m\_L1D$ 根据约束条件(3)取最大整数值.

根据上述分块参数设计方法,应用到我们的目标测试处理器 MATRIX2 上,得到最优的子块矩阵参数如下:

(1)矩阵 $\boldsymbol{A}$ 的子块 $m\_L1D\times k\_gc$:6×512;

(2)矩阵 $\boldsymbol{B}$、$\boldsymbol{C}$ 的子块 $k\_gc\times n\_am$:512×48;

(3)矩阵 $\boldsymbol{A}$ 在 GC 中的子块 $m\_gc\times k\_gc$:512×512.

其中最内层寄存器级的核心循环计算子块参数是:$\boldsymbol{A}$ 子块:6×512,$\boldsymbol{B}$ 子块:512×48,$\boldsymbol{C}$ 的子块:6×48.核心循环的软件流水表如图 10,软件流水后标量数据的加载、广播与向量数据加载、乘加计算并行处理,不会影响核心计算.其中向量乘加指令开销为 6 拍,循环填充和排空开销为 19 拍,实际的核心计算开销为 512×6+19=3091 拍,计算效率为 99.39%,非常接近处理器的峰值计算效率.

| 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|
| SLDW | SLDW | SLDW | SLDW | SLDW | SLDW |
| VLDDW | VLDDW | VLDDW | VLDDW | VLDDW | VLDDW |
| VLDW | VLDW | VLDW | VLDW | VLDW | VLDW |
| SVBCAST | SVBCAST | SVBCAST | SVBCAST | SVBCAST | SVBCAST |
| VFMULAD1 | VFMULAD4 | VFMULAD7 | VFMULAD10 | VFMULAD13 | VFMULAD16 |
| VFMULAD2 | VFMULAD5 | VFMULAD8 | VFMULAD11 | VFMULAD14 | VFMULAD17 |
| VFMULAD3 | VFMULAD6 | VFMULAD9 | VFMULAD12 | VFMULAD15 | VFMULAD18 |
| SBR | | | | | |
| SUB | | | | | |

图 10 核心循环的软件流水表

### 4.6 计算访存比和数据带宽分析

矩阵乘法是典型的计算和访存密集型算法,衡量算法效率高低的一个重要参考指标是运算访存比,除了可以量化参数的影响外,还可以反映对存储带宽的需求.从图 8 中可以看到,分块矩阵乘法计算中的多级存储间的数据搬移主要有 3 种情况:

(1)矩阵 $\boldsymbol{A}$ 的子块已经缓存在 GC 中时,矩阵 $\boldsymbol{B}$、$\boldsymbol{C}$ 的子块 $k\_gc\times n\_am$ 在 DDR 和 AM 之间的数据搬移.假设矩阵 $\boldsymbol{A}$ 的子块 $m\_gc\times k\_gc$ 一直驻留在 GC 中,直到不再需要为止;矩阵 $\boldsymbol{B}$、$\boldsymbol{C}$ 的子块 $k\_gc\times n\_am$ 循环搬移至 AM 的双缓冲中,则计算访存比计算如下:

$$f_1=\frac{flops}{memops}=\frac{2\times m\_gc\times k\_gc\times n\_core}{m\_gc\times k\_gc+(k\_gc+2\times m\_gc)\times n\_core}.$$

(2)矩阵 $\boldsymbol{B}$、$\boldsymbol{C}$ 的子块已经搬移至 AM 的缓冲区中,矩阵 $\boldsymbol{A}$ 的子块 $m\_L1D\times k\_gc$ 在 GC 和 L1D SRAM 之间的数据搬移.假设矩阵 $\boldsymbol{B}$、$\boldsymbol{C}$ 的子块 $k\_gc\times n\_am$ 驻留在 AM 的缓冲区中,矩阵 $\boldsymbol{A}$ 的子块 $m\_gc\times k\_gc$ 一直驻留在 GC 中,直到不再需要为止,矩阵 $\boldsymbol{A}$ 的子块 $m\_L1D\times k\_gc$ 循环搬移至 SRAM 的双缓冲中,则运算访存比计算如下:

$$f_2=\frac{flops}{memops}=\frac{2\times m\_gc\times k\_gc\times n\_am}{m\_gc\times k\_gc+(k\_gc+2\times m\_gc)\times n\_am}.$$

(3)矩阵 $\boldsymbol{B}$、$\boldsymbol{C}$ 的子块已经搬移至 AM 的缓冲区,矩阵 $\boldsymbol{A}$ 的子块已经搬移至 L1D SRAM 中,此时核心计算是以处理器峰值速度进行,则运算访存比计算如下:

$$f_3=\frac{flops}{memops}=\frac{2\times m\_L1D\times k\_gc\times n\_am}{m\_L1D\times k\_gc+(k\_gc+2\times m\_L1D)\times n\_am}.$$

根据这 3 个运算访存比计算公式,结合图 4 所示的基于两级 DMA 双缓冲的优化数据搬移策略,提高 GEMM 计算性能的分块参数优化设计采用如

下方法：

(1)矩阵 $\boldsymbol{A}$ 的子块 $m\_gc\times k\_gc$ 根据 GC 的容量约束最大化，矩阵 $\boldsymbol{A}$ 的子块 $m\_gc\times k\_gc$ 驻留在 GC 中，直到 $\boldsymbol{B}$、$\boldsymbol{C}$ 在 $n$ 维方向的全部子块全部计算完毕；

(2)矩阵 $\boldsymbol{A}$ 的子块 $m\_L1D\times k\_gc$ 根据 SRAM 的容量约束最大化，矩阵 $\boldsymbol{A}$ 的子块 $m\_L1D\times k\_gc$ 驻留在 SRAM 中，直到 $\boldsymbol{B}$、$\boldsymbol{C}$ 的子块 $k\_gc\times n\_am$ 的计算完毕.

影响处理器计算效率的另一个关键因素是存储访问延迟问题，为保证核心计算的高效率，数据传输带宽必须保证数据在计算需要时就绪. 我们提出的分块矩阵乘法计算中通过两次 DMA 双缓冲方法实现数据搬移与计算重叠，能否真正的隐藏数据传输时间，取决于设计的分块参数能否使数据带宽满足硬件提供的约束条件.

(1)矩阵 $\boldsymbol{A}$ 的子块数据链路带宽要求. 矩阵 $\boldsymbol{A}$ 的子块 $m\_L1D\times k\_gc$ 与矩阵 $\boldsymbol{B}$、$\boldsymbol{C}$ 的子块 $k\_gc\times n\_am$ 的核心计算中，DMA 搬移的数据为 $\boldsymbol{A}$ 的子块 $m\_L1D\times k\_gc$，设 $t_k$ 为该子块以峰值性能的计算时间，$P$ 为多核数量，则 GC 至 L1D SRAM 的数据带宽 $R_1$ 必须满足要求：

$$R_1\geq\frac{P\times m\_L1D\times k\_gc}{t_k}.$$

(2)矩阵 $\boldsymbol{B}$、$\boldsymbol{C}$ 的子块数据链路带宽要求. 在满足条件(1)的情况下，矩阵 $\boldsymbol{A}$ 的子块 $m\_gc\times k\_gc$ 与矩阵 $\boldsymbol{B}$、$\boldsymbol{C}$ 的子块 $k\_gc\times n\_am$ 的计算中，需要 DMA 搬入 $\boldsymbol{B}$、$\boldsymbol{C}$ 的子块数据各 1 次，搬出 $\boldsymbol{C}$ 的子块数据 1 次，则 DDR 至 AM 的数据带宽 $R_2$ 必须满足要求：

$$R_2\geq\frac{P\times(k\_gc\times n\_am+2\times m\_gc\times n\_am)}{(m\_gc/m\_L1D)\times t_k}.$$

在 MATRIX2 平台的实际测试中，上述数据带宽 $R_1$、$R_2$ 均满足要求，即 4.3 节提出的基于两级 DMA 双缓冲的数据搬移优化策略能够将数据搬移开销隐藏，从而有效保证了本文提出的 GEMM 的计算效率.

## 5　性能测试与分析

首先，我们在 MATRIX2 平台上测试了不同分块参数下，单核的最小核心 GEPB 的双精度浮点计算性能和效率，如表 1 所示.

**表 1　不同分块下核心 GEPB 的双精度浮点计算性能和效率**

| 矩阵 $\boldsymbol{A}$ $m\_L1D\times k\_gc$ | 矩阵 $\boldsymbol{B}$ $k\_gc\times n\_am$ | 矩阵 $\boldsymbol{C}$ $m\_gc\times n\_am$ | kernel GEPB 性能 (GFLOPS) | kernel GEPB 计算效率 |
|---|---|---|---|---|
| 2×2048 | 2048×16 | 512×16 | 91.72 | 95.54% |
| 4×1024 | 1024×16 | 1024×16 | 88.02 | 91.69% |
| 4×1024 | 1024×32 | 512×32 | 91.92 | 95.75% |
| 6×512 | 512×48 | 512×48 | 95.41 | 99.39% |

从表 1 可以看出，各种分块参数满足 MATRIX2 的约束条件，其中 $n\_am$ 取参数 $v\times c=16\times 3=48$ 时，核心 GEPB 的计算性能最高，达 95.41 GFLOPS，非常接近单核的峰值性能 96 GFLOPS，其计算效率也高达 99.39%. 测试结果也表明 4.6 节提出的最优分块参数设计方法是有效的. 同时也表明基本块 NB 设定为 512 时最为合适. 根据前面分块 LU 求解过程，无论是 $\boldsymbol{L}$、$\boldsymbol{U}$ 矩阵更新，还是尾矩阵更新，都是 $m$，$n$ 方向较长，$k$ 方向设定基本块 512 较为合适，后续性能测试根据 $m$，$n$ 方向做数据规模扩展测试.

如图 11 所示，选定矩阵 $\boldsymbol{A}$ 在 GC 中的子块参数为 512×512，我们进一步测试不同矩阵规模下的双精度 GEMM 性能. 测试中矩阵 $\boldsymbol{A}$ 的规模为 512×512，矩阵 $\boldsymbol{B}$、$\boldsymbol{C}$ 的规模依据 $q$ 值变化为 $512\times 48\times q$. 图 11 给出了不同规模($q$ 值)下单核的双精度 GEMM 的计算性能和效率测试结果. 图的左侧纵坐标是计算性能(GFLOPS)，用于标示图中直方图显示的 GEMM 计算性能. 右侧纵坐标是计算效率用于标示图中折线图显示的 GEMM 计算效率. 从图中可以看到，当 $q$ 值较小时，单核的双精度 GEMM 计算效率只有 85.92%. 这是因为在 4.4 节中提出的基于两级 DMA 双缓冲的优化数据搬移策略中，尽管在计算的过程中，两个缓冲区的计算与 DMA 的数据搬移重叠，但是有 3 次 DMA 的数据搬移时间是不能隐藏的，包括第 1 次矩阵 $\boldsymbol{A}$ 的子块 $m\_gc\times k\_gc$ 从 DDR 搬入至 GC 中、第 1 次矩阵 $\boldsymbol{B}$、$\boldsymbol{C}$ 的子块 $k\_gc\times n\_am$ 从 DDR 搬入至 AM 中以及最后 1 次矩阵 $\boldsymbol{C}$ 的计算结果子块 $k\_gc\times n\_am$ 从 AM 搬出至 DDR. 在矩阵规模较小时，这三次 DMA 搬移时间在整个 GEMM 的总执行时间占比较大，例如 $q=8$ 时，三次 DMA 搬移时间在整个 GEMM 总执行时间占比约为 11.2%，导致 GEMM 的计算效率只有 85.92%. 随着矩阵规模的逐步扩大，当 $q=1024$ 时，三次 DMA 搬移时间被平摊，在整个 GEMM 总执行时间占比很小，使得 GEMM 的计算性能达到 92.88 GFLOPS，非常接近单核的峰值性能 96 GFLOPS，计算效率达到 96.75%.

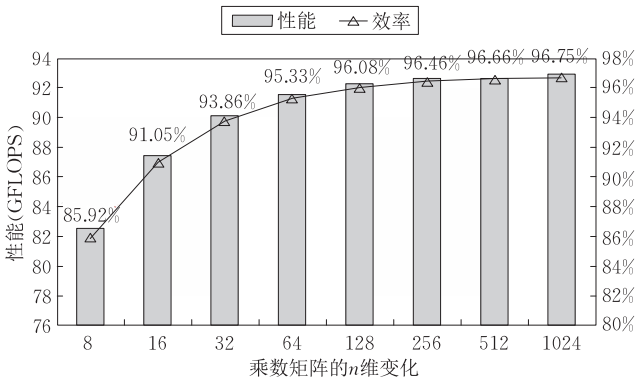



图 11 单核的双精度 GEMM 计算性能和效率

如图 12 所示,同样选定矩阵 $\boldsymbol{A}$ 在 GC 中的子块参数为 512×512,我们在 MATRIX2 上进一步测试大矩阵的多核并行双精度 GEMM 的性能和效率.测试中 $m$、$k$ 和 $n$ 依据 $r$ 值变化为 $m=k=512\times r$,$n=48\times 12\times r$.图 12 给出了不同矩阵规模($r$ 值)下多核的双精度 GEMM 的计算性能和效率测试结果.从图中可以看到,当 $r$ 值较小时,多核的双精度 GEMM 计算效率只有 47.99%,这与图 11 中展示的单核测试结果一致,因为有 3 次 DMA 的数据搬移时间是不能隐藏的,在矩阵规模较小时,在整个 GEMM 的总执行时间占比较大.随着矩阵规模的逐步扩大,GEMM 的性能稳步上升,当 $r=128$ 时,GEMM 的计算性能达到 1106.88 GFLOPS,非常接近多核的峰值性能 1152 GFLOPS(图中直方图),计算效率达到 96.08%(图中折线图的三角标记).

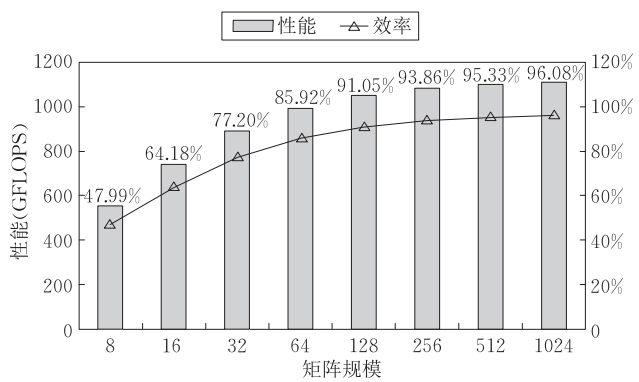



图 12 多核的双精度 GEMM 计算性能和效率

图 13 对比了 MATRIX2 的多核 GEMM、Intel CPU 平台上的 MKL①、NVIDIA GPU 平台上的 cuBLAS②以及 TI C66x 平台的测试结果[2].其中 MATRIX2_GEMM、cuBLAS、MKL 和 TI_BLAS 分别表示在 MATRIX2(12 核,单精度浮点峰值 2.3TFLOPS)、Tesla M40(3072 核,单精度浮点峰值 6.84TFLOPS)、Intel Xeon Phi5110P(60 核,单精度浮点峰值 2.02TFLOPS)和 TI C66x(8 核,单精度浮点峰值 128 GFLOPS)平台上报告的单精度 GEMM 性能和效率(对应的数据选择引用文献上最好的结果).从图中可以看出,MATRIX2_GEMM 取得的性能和效率显著高于其他算法库.

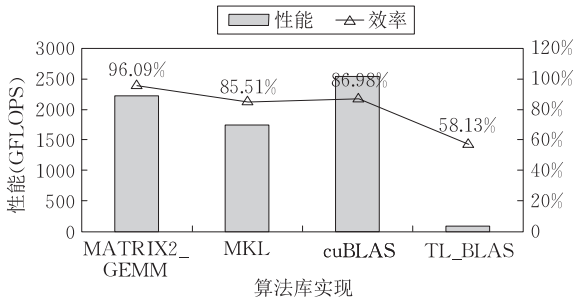



图 13 不同平台算法库单精度 GEMM 性能和效率

从功耗情况看,MATRIX2 的单芯片功耗是 160 W,测试得到的单精度 GEMM 性能约为 2214 GFLOPS.根据文献[12]的数据,表 2 对比了不同平台上单精度 GEMM 的性能、单位功耗性能(实测性能/功耗)和效率.从单位功耗性能数值上看,CPU 数据最差,DSP 优于 GPU,MATRIX2 最好,显示 DSP 在高性能计算方面具有明显的功耗优势,而 MATRIX2 是一款基于 VLIW 的多核向量处理器,在强化功耗优势的同时,在性能、效率也极具优势,适合高性能计算.但是也面临着如何根据其体系结构特点高效的并行和向量化应用程序的困难,目前缺乏高效的并行和向量化编译软件开发平台.

**表 2 不同平台上的单精度 GEMM 的性能、功耗性能和效率**

| 处理器 | GFLOPS | GFLOPS/W | 效率 |
|---|---|---|---|
| Core i7-960 | 96 | 1.14 | 95% |
| Nvidia GTX280 | 410 | 2.6 | 66% |
| Cell | 200 | 5.0 | 88% |
| Nvidia GTX480 | 940 | 5.4 | 70% |
| Stratix IV | 200 | 7.0 | 90+% |
| TI C66x DSP | 74 | 7.4 | 57% |
| MATRIX2 | 2214 | 13.83 | 96% |

## 6 结 论

传统的分块矩阵乘法方法针对基于 Cache 的多级存储结构,将大矩阵的乘法分割为一系列子矩阵的乘法,子矩阵的分块参数通常依据 Cache 的容量进行优化设计,尽量让各级子矩阵计算所需的数据

① https://www.Intel.com/content/www/us/en/benchmarks/server/xeon-phi/xeon-phi-competitive-performance.html
② https://www.acceleware.com/blog/tesla-meets-maxwell

访问能够在 Cache 中命中，使得该部分的计算能以接近峰值的性能进行，从而提高整个大矩阵乘法的计算性能. 本论文针对多核向量处理器的体系结构特点，提出一种基于 SRAM 的 GEMM 向量化优化方法，设计了两级 DMA 双缓冲的优化数据搬移策略，通过优化分块参数设计，使得数据搬移时间隐藏于计算时间，依据向量处理器体系结构特点设计最优的子矩阵块参数和计算核心，使得整个 GEMM 计算以接近处理器峰值的性能进行，大幅度提高处理器的计算性能和效率. 同时，本文对提出的分块参数和计算核心的优化设计方法给出了定量分析和计算，在 MATRIX2 核数量或者每个核的 PE 数量发生变化时仍然具有通用性；提出的按行计算的矩阵乘法向量化方法对于类似 HPL 尾矩阵更新的通用稠密矩阵乘法计算，采用共享矩阵 $\boldsymbol{A}$，具有减轻 DDR 访存压力，避免通常向量处理器不支持的对乘数矩阵列向量数据的访问，以及硬件开销较大的向量处理单元之间的浮点求和归约设计具有参考意义. 在 MATRIX2 上的实验测试结果表明，本文提出的矩阵乘法向量化方法能够取得比传统方法更高的计算性能和效率.

## 参　考　文　献


[1] Chi Xue-Bin, Gu Bei-Bei, Wu Hong, et al. The analysis of development on high performance computer system and platform. Computer Engineering & Science, 2013, 35(11): 6-13(in Chinese)
(迟学斌，顾蓓蓓，武虹等. 高性能计算机系统及平台发展状况分析. 计算机工程与科学，2013，35(11)：6-13)

[2] Igual F D, Ali M, Friedmann A, Stotzer E, et al. Unleashing DSPs for general-purpose. HPC. FLAME Working Note # 61. Technical Report TR-12-02, The University of Texas at Austin, Department of Computer Sciences, Austin, USA, 2012

[3] Liu Zhong, Chen Yue-Yue, Chen Hai-Yan. A vectorization of FIR filter supporting arbitrary coefficients length and data types. Acta Electronica Sinica, 2013, 41(2): 346-351(in Chinese)
(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. 电子学报，2013，41(2)：346-351)

[4] Goto K, van De Geijn R. High-performance implementation of the level-3 BLAS. ACM Transactions on Mathematical Software, 2008, 35(1): 1-18

[5] Zhang Xian-Yi, Wang Qian, Zhang Yun-Quan. OpenBLAS: A high performance blas library on loongson 3A CPU. Journal of Software, 2011, 22(Supplement(2)): 208-216(in Chinese)
(张先轶，王茜，张云泉. OpenBLAS：龙芯 3A CPU 的高性能 BLAS 库. 软件学报，2011，22(增刊(2))：208-216)

[6] Sohl J, Wang J, Liu D. Large matrix multiplication on a novel heterogeneous parallel DSP architecture//Proceedings of the 8th International Symposium on Advanced Parallel Processing Technologies. Rapperswil, Switzerland: Springer, 2009: 408-419

[7] Zhang Kai. High Efficient Matrix Operations on Vector-SIMD DSPs [Ph. D. dissertation]. National Defense University of Science and Technology. Changsha, 2013(in Chinese)
(张凯. 向量 SIMD DSP 上高效矩阵运算技术研究[博士学位论文]. 国防科学技术大学，长沙，2013)

[8] Chen Shao-Hu, Zhang Yun-Quan, Zhang Xian-Yi, Cheng Hao. Performance testing and analysis of BLAS libraries on multi-core CPUs. Journal of Software, 2010, 21(Supplement): 214-223(in Chinese)
(陈少虎，张云泉，张先轶，程豪. BLAS 库在多核处理器上的性能测试与分析. 软件学报，2010，21(增刊)：214-223)

[9] Gunnels J A, Henry G M, van de Geijn R A. A family of high-performance matrix multiplication algorithms//Proceedings of the International Conference on Computational Science Part I. London, UK, 2001: 51-60

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

[11] Eric Stotzer M A, Igual F D, van de Geijn R A. Level-3 BLAS on the TI C6678 multi-core DSP//Proceedings of the IEEE 24th International Symposium on Computer Architecture and High Performance Computing. New York, USA, 2012: 179-186

[12] Pedram A, van de Geijn R, Gerstlauer A. Co-design tradeoffs for high-performance, low-power linear algebra architectures. IEEE Transactions on Computers, 2012, 61(12): 1724-1736



**LIU Zhong**, born in 1971, Ph. D., professor. His main research interests include high performance computing, parallel computing and performance optimization.

**TIAN Xi**, born in 1986, M. S. candidate. His main research interests include high performance computing, analysis of the algorithm and the designing.

**Background**

Tianhe-2 is the world's most powerful publicly known supercomputer, which uses a mix of E5-2692 CPUs and Xeon Phi accelerators. It emerged that Uncle Sam had forbidden Intel from shipping high-end Xeon and Xeon Phi parts to China's defense labs and other areas of its supercomputing industry in April 2015. Essentially, the next generation supercomputer will use the China-crafted new accelerator instead of the Xeon Phi accelerators; the new accelerator for the upcoming supercomputer was referred to as both the "China Accelerator" and GPDSP. GPDSP cores employ 11-issue VLIW and scalar-vector coupled architecture with both scalar and vector units, dedicated vector memory, and VLIW capabilities. Both the data path and memory hierarchy of the GPDSP architecture are highly optimized for scientific computing such as the large scale matrix computation.

Dense matrix multiplication is one of the core computations in many algorithms from large scientific computing and its efficiency impacts performances of almost all matrix problems. In order to fully exploit the potential of such VLIW+SIMD architectures for high performance computing, we systematically study and evaluate the performances of dense general matrix-matrix multiplication algorithm on the architecture. It is a part of project on automatic vectorization code generation and performance tuning techniques based on tensor product. This project is supported by the National Natural Science Foundation of China under Grant Nos. 61572025, 61472432.

This paper presented an efficient vectorization of matrix multiplication for multi-core vector processors, which included a vectorization of matrix multiplication according to row computation and an optimized matrix blocking method. Experimental results show that the performance of presented double precision matrix multiplication achieves 1106.88 GFLOPS, an efficiency of 96.08%, and the efficiency of kernel computation achieves 99.39%.