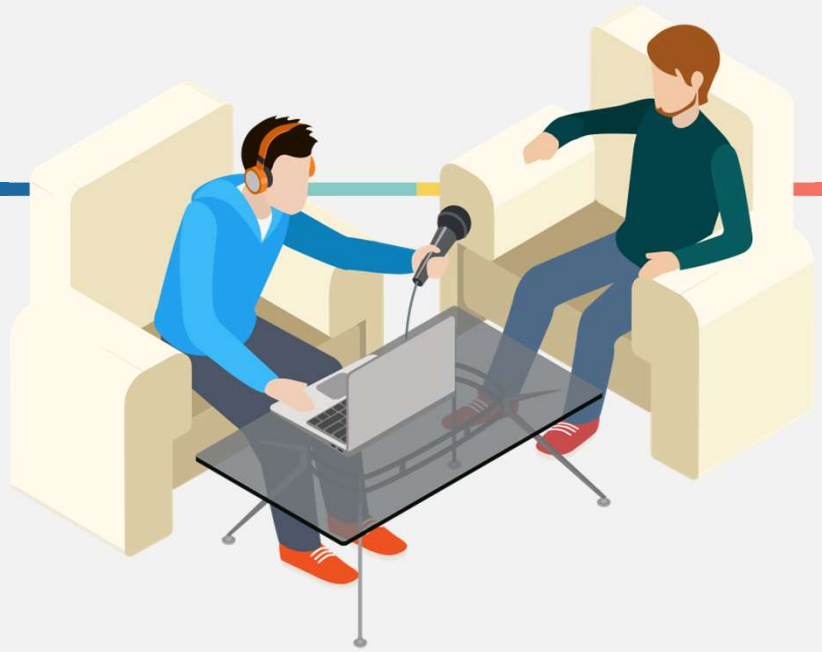
64位ARMv8多核处理器高效DGEMM的设计与实现

Design and Implementation of a Highly Efficient DGEMM for 64-bit ARMv8 Multi-Core Processors

汇报人：刘宜佳、郭凌超、王东紫
指导老师：龚春叶、甘新标、杨博

目 录

CONTENTS



01 介绍

- ✓ 选题背景
- ✓ 实现方法
- ✓ 主要挑战

介绍

选题背景：

- 基于ARMv8的SoC构建HPC系统；
- 双精度通用矩阵乘法（**DGEMM**） 一直是衡量HPC平台性能的重要核心；
- **稠密矩阵运算**在科学和工程计算中发挥着重要作用。

基本线性代数子程序（**BLAS**）规定了用于发布库的应用程序编程接口标准；
对于Level3 BLAS， 最常用的**矩阵-矩阵**计算可以实现为一般矩阵乘法。

实现方法：

采用理论指导的方法， 首先为这个架构开发一个性能模型， 然后用它来指导我们的探索。

主要挑战：

选择正确的最内层寄存器内核的**寄存器块大小**。

目标是最大化从L1数据缓存到寄存器的**计算到内存的访问比率**。

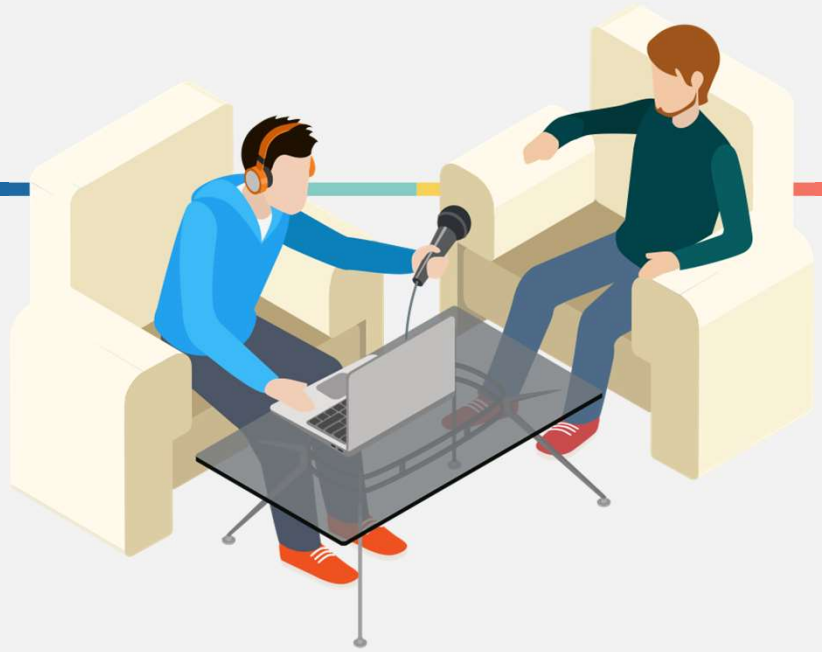
介绍

实现最佳计算内存比率

01 利用循环展开，指令调度和软件实现的寄存器旋转；

02 利用A64指令实现高效的FMA（乘加）操作，数据传输和数据预取；

03 通过分析确定GEBP（block-panel乘法）使用的剩余性能关键块大小，优化GEBP， 最大化它的计算到内存访问比率。

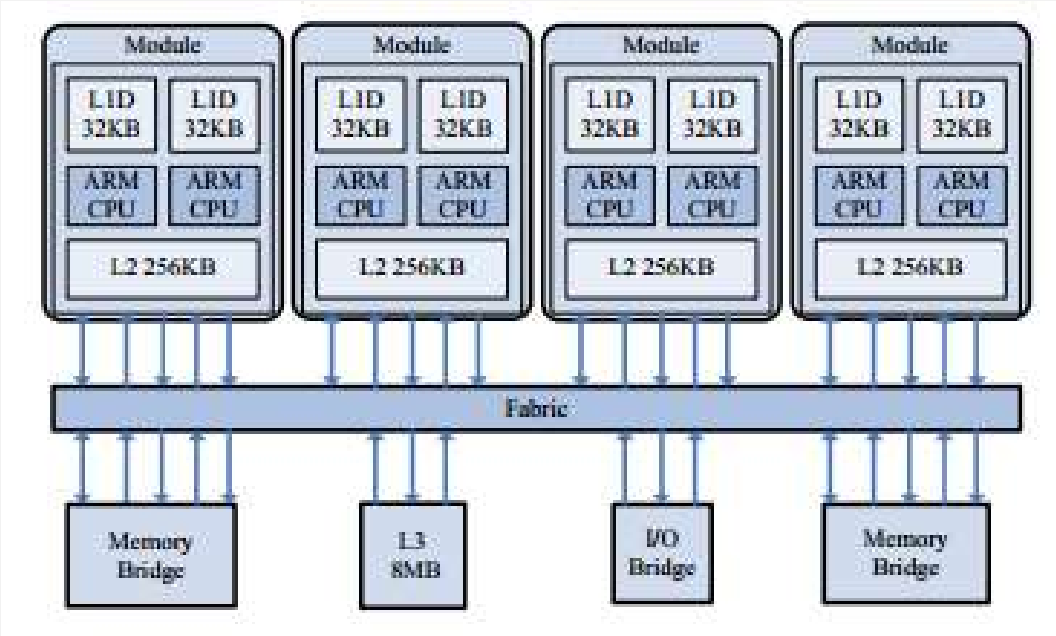


02 背景

- ✓ 64位ARMv8多核处理器
- ✓ 开源BLAS实现
- ✓ OpenBLAS中的DGEMM概述

背景

64位ARMv8多核处理器



64位ARMv8八核处理器

- 32KB的L1指令缓存和32KB的L1数据缓存；
- 同一模块中的两个内核共享256 KBL2高速缓存，四个模块（八个内核）共享一个8MB L3高速缓存；
- 每个内核都有一个支持FMA的浮点计算流水线，运行频率为2.4GHz， 峰值性能为4.8Gflops；
- 每个核有32个128位浮点寄存器v0-v31，可用于浮点向量计算。

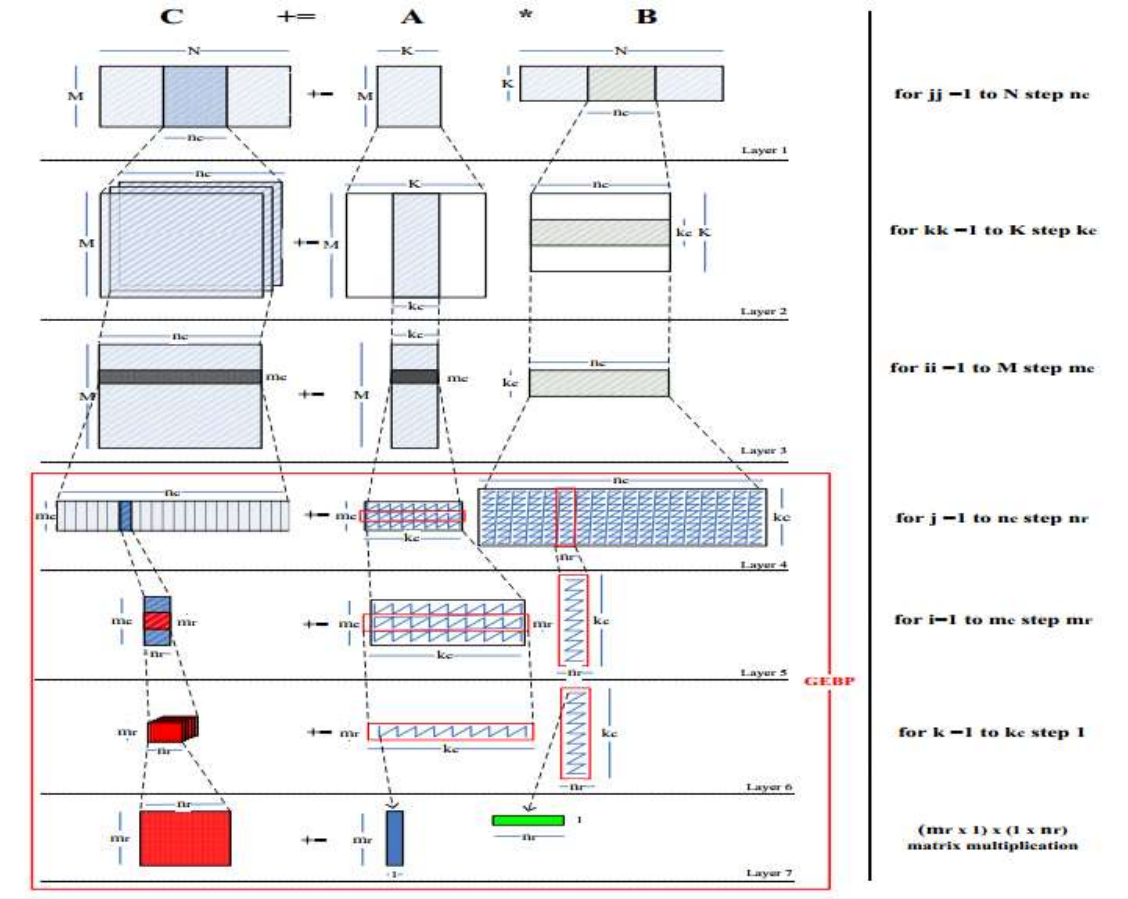
背景

开源BLAS实现

- 几个着名的开源BLAS：netlib BLAS，ATLAS,GotoBLAS,OpenBLAS和BLIS；
- 在OpenBLAS中，其内核称为GEBP（计算的基本单位），通常用汇编中实现。

背景

OpenBLAS中的DGEMM概述

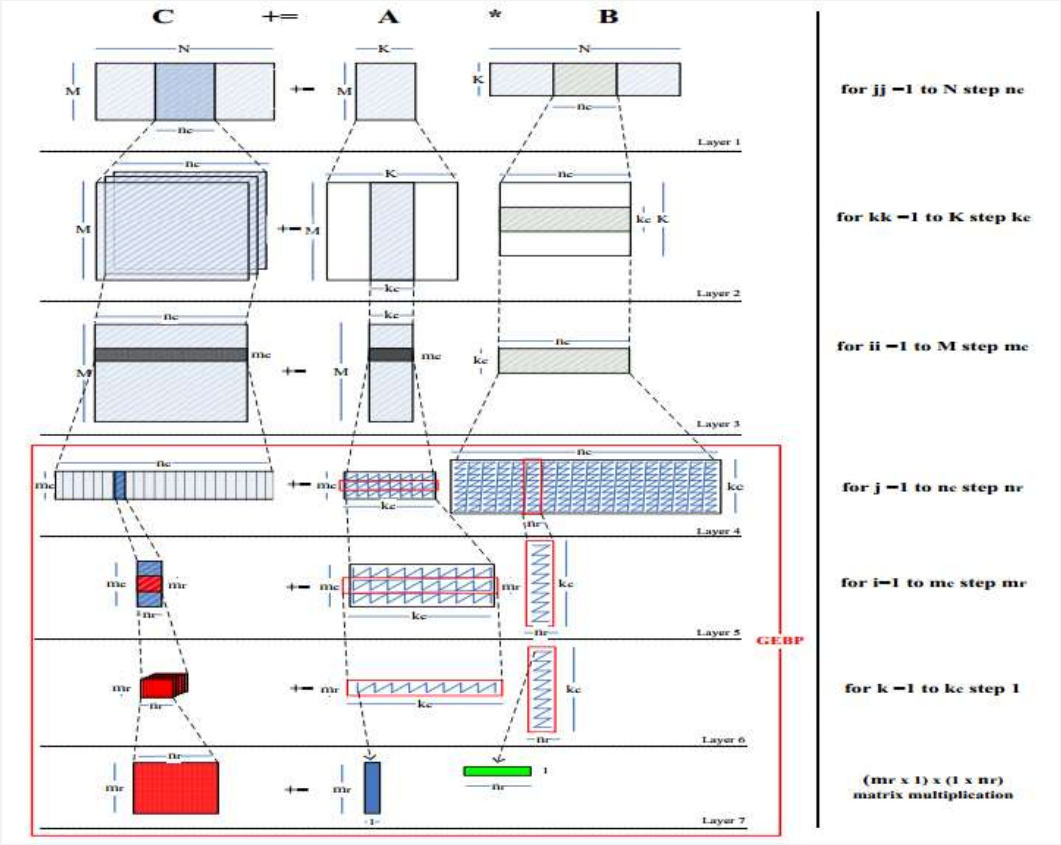


DGEMM算法实现

- 矩阵相乘：C + = AB;
- blocking（最大化缓存性能）和packing（使数据有效地移动到寄存器）。

背景

OpenBLAS中的DGEMM概述



层1

将C和B分别分成大小为M×nc和K×nc的列面板

层2

将M×K矩阵A和K×nc B的子矩阵分别分成M×Kc列面板和kc×nc的行面板

层3

A的每个M×Kc被分成mc×kc块

层4

为了确保连续访问， OpenBLAS将A（B）的块（面板）打包成连续的缓冲区

层5

GEBS：将B的kc×nc面板划分为kc×nr条

层6

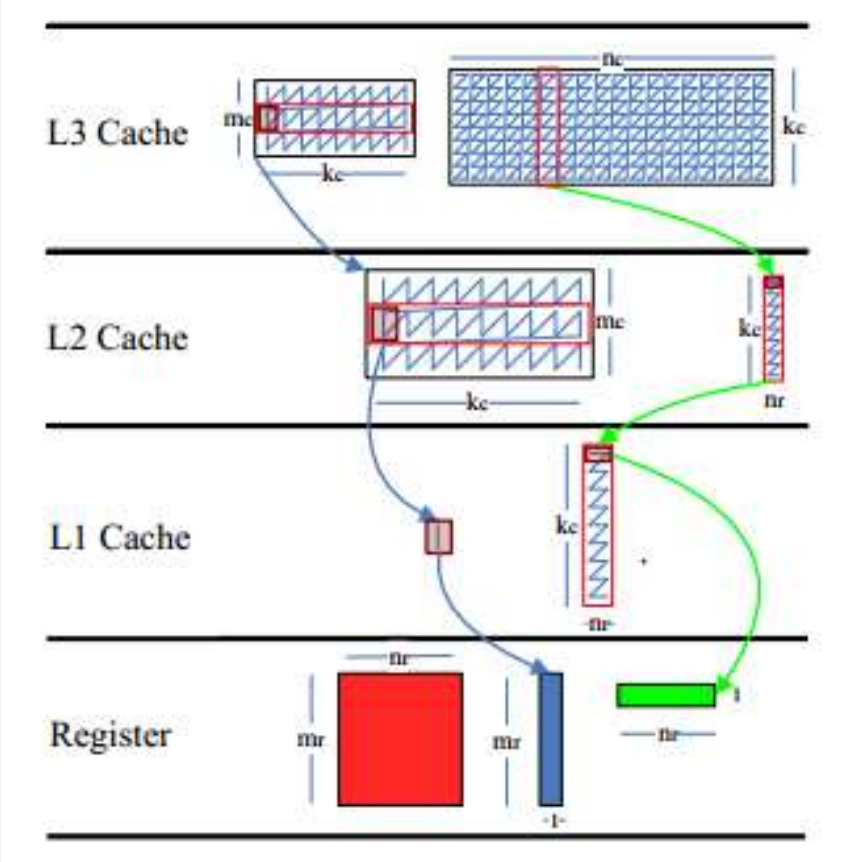
GESS：将A的mc×kc块分割成mr×kc条

层7

矩阵乘法：（mr x 1）x（1 x nr）

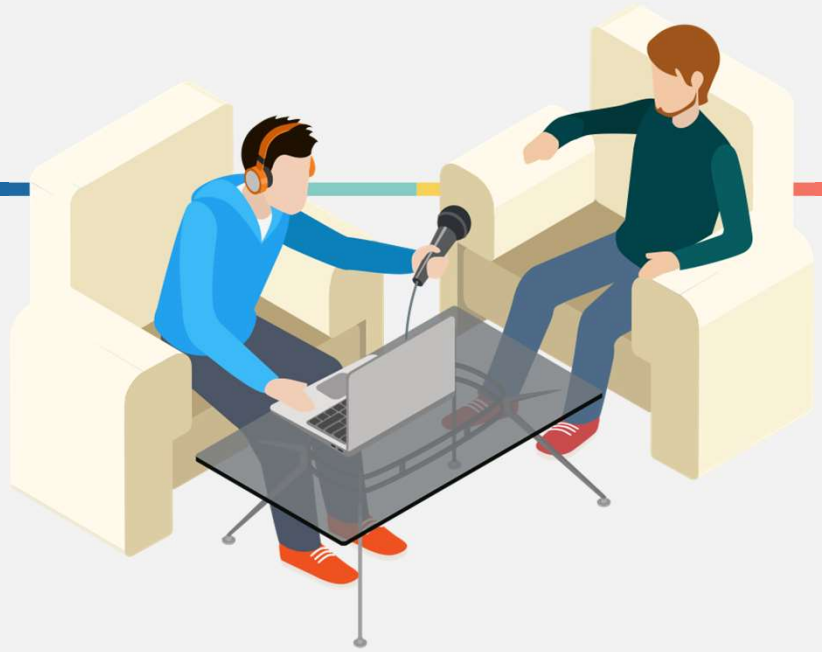
背景

OpenBLAS中的DGEMM概述



GEBP打包数据存储

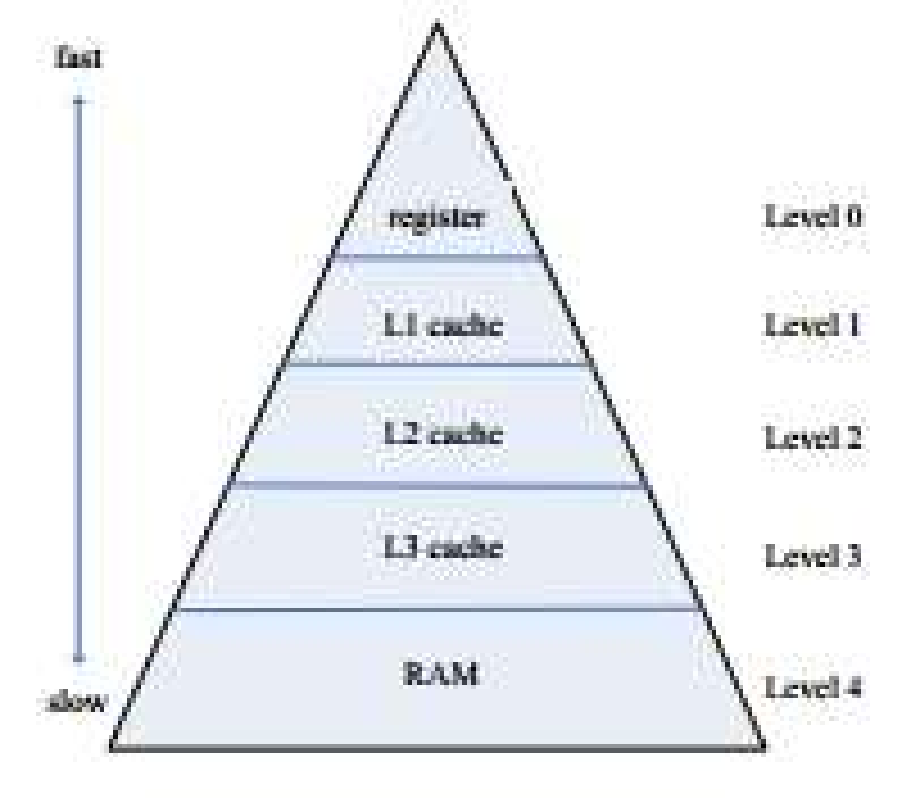
- 打包A：提取一系列大小为mc×kc的条（子块），来自A的mc×kc块；并且在L2 cache中组织这些条子。
- 打包B：提取一系列大小为kc×nr的条子，来自B的kc×nc面板，并且在L3 cache中组织这些条子。
- 最后一层称为寄存器内核。



04 性能建模

- ✓ 理论基础
- ✓ **DGEMM**性能下限计算

性能建模



ARMv8架构内存层次

- **理论基础：**关于CPU速度与内存速度之比（**即计算内存访问比**）的通用性能模型。
- 如左图所示，必须注意有效地处理从内存到寄存器的**数据移动量**和**数据计算量**。
- 执行时间：

$$T = F\mu + \sum_i \sum_j W_{ij}\nu_{ij} + \sum_i \sum_j M_{ij}\eta_{ij} \qquad (1)$$

vij :将一个字（浮点值）从层i移动到层j所需时间；
nij:将一个消息（由连续字组成的cache行）从层i移动到层j所需时间；
F， Wij和Mij分别代表操作，字和消息的数量。

性能建模

DGEMM性能下限计算

移动消息数和移动字数之比大约是常数：

$$\sum_i \sum_j M_{ij} \simeq \kappa \sum_i \sum_j W_{ij}$$

由于 vij >= 0 且 nij >= 0

所以执行时间T：

$$T \leq F\mu + (1+\kappa) \sum_i \sum_j W_{ij} \times \left(\sum_i \sum_j \nu_{ij} + \sum_i \sum_j \eta_{ij} \right)$$

性能建模

DGEMM性能下限计算

为方便起见，让：

$$\pi = \sum_i \sum_j \nu_{ij} + \sum_i \sum_j \eta_{ij} \qquad W = \sum_i \sum_j W_{ij}$$

执行时间T：

$$T \leq F\mu + (1+\kappa)W\pi \tag{3}$$

计算内存访问比：

$$\gamma = \frac{F}{W} = \frac{F}{\sum_i \sum_j W_{ij}} \tag{2}$$

性能建模

DGEMM性能下限计算

重叠因子ψ（γ），优化后的T：

$$T_{opt} \leq F\mu + (1+\kappa)W\pi\psi(\gamma) \quad (4)$$

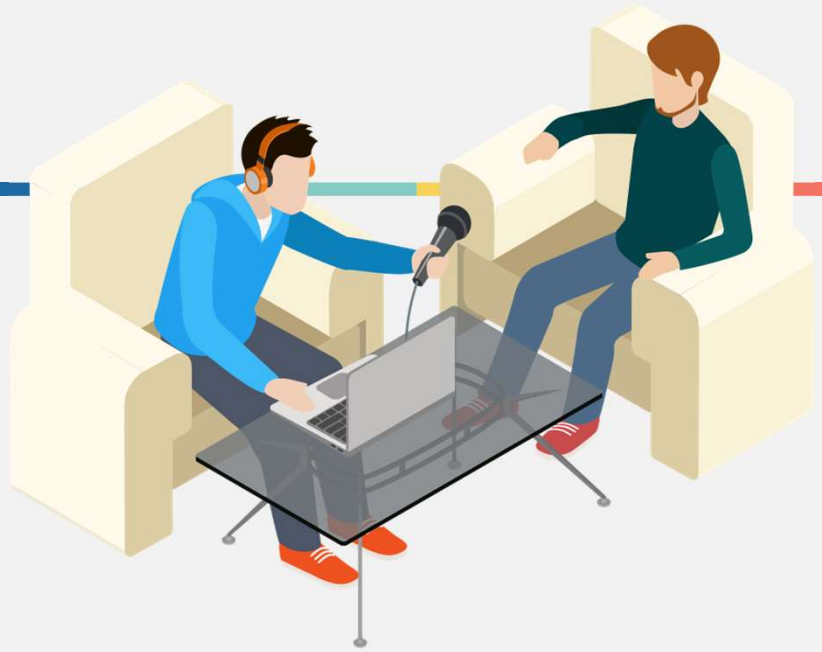
ψ（γ）是γ的单调递减函数，通过（2）有：

$$T_{opt} \leq F(\mu + (1+\kappa)\pi\frac{\psi(\gamma)}{\gamma}) \quad (5)$$

最终，DGEMM性能下限：

$$Perf_{opt} = \frac{F}{T_{opt}} \geq \frac{1}{(\mu + (1+\kappa)\pi\frac{\psi(\gamma)}{\gamma})} \quad (6)$$

→ 清楚地表明了更大的计算内存比率γ能达到更好的峰值性能（效率）



05 快速实现

- ✓ 寄存器分块
- ✓ 局部缓存分块
- ✓ 并行缓存分块

快速实现

主要挑战

选择正确的内核寄存器大小

目标

最大化从L1数据缓存到寄存器的计算到内存的访问比率

实现最佳计算内存比率

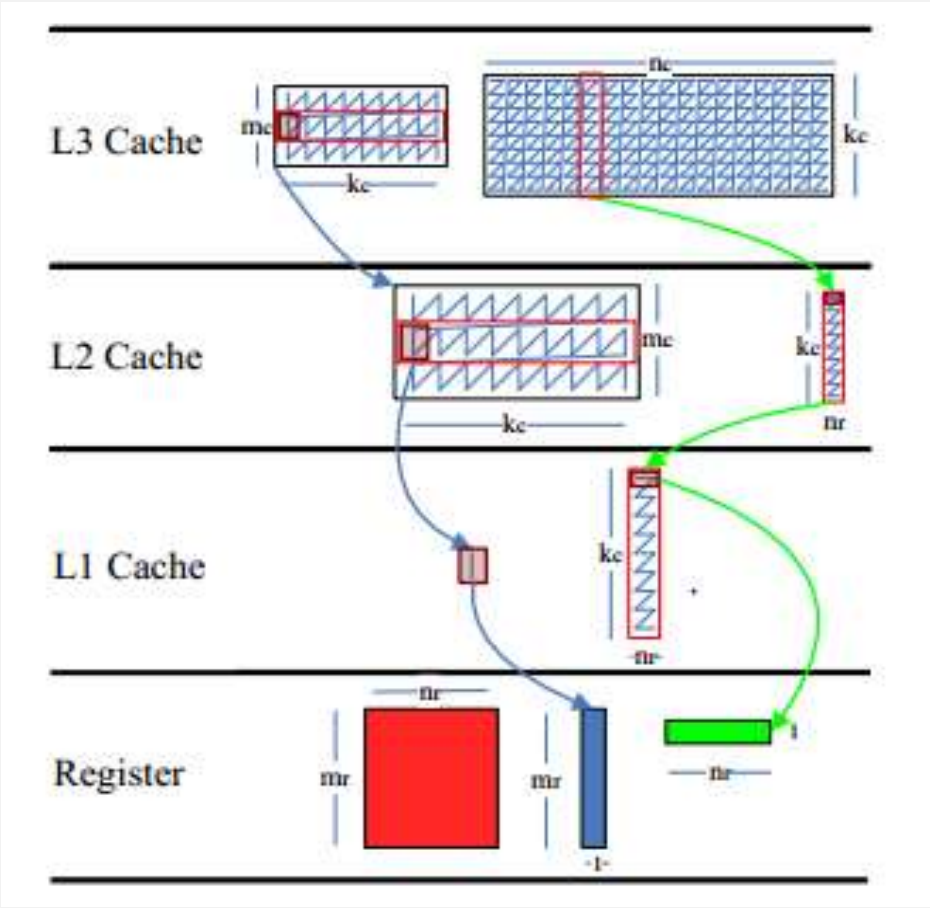
01 利用循环展开，指令调度和软件实现的寄存器旋转；

02 利用A64指令实现高效的FMA（乘加）操作，数据传输和数据预取；

03 通过分析确定GEBP（block-panel乘法）使用的剩余性能关键块大小，优化GEBP， 最大化它的计算到内存访问比率。

快速实现

寄存器分块



计算内存访问比：

$$\frac{2m_r n_r}{(m_r \times 1)_{L1\to R} + (n_r \times 1)_{L1\to R}} \tag{7}$$

优化问题：

$$\max \quad \gamma = \frac{2}{\frac{1}{n_r} + \frac{1}{m_r}} \tag{8}$$

快速实现

寄存器分块

约束条件：

01

$$(m_r n_r + 2m_r + 2n_r) \times element\_size \le (n_f + n_{rf}) \times p_f \quad (9)$$

element_size：以字节为单位的矩阵元素的大小

nf：可用的浮点寄存器数量

nrf：可以重用于预加载的寄存器数量

pf：以字节为单位的浮点寄存器的大小

02

$$0 \le n_{rf} \times p_f \le (m_r + n_r) \times element\_size. \quad (10)$$

03

$$m_r = 2i,\ n_r = 2j,\ i = 1, 2, \cdots,\ j = 1, 2, \cdots \quad (11)$$

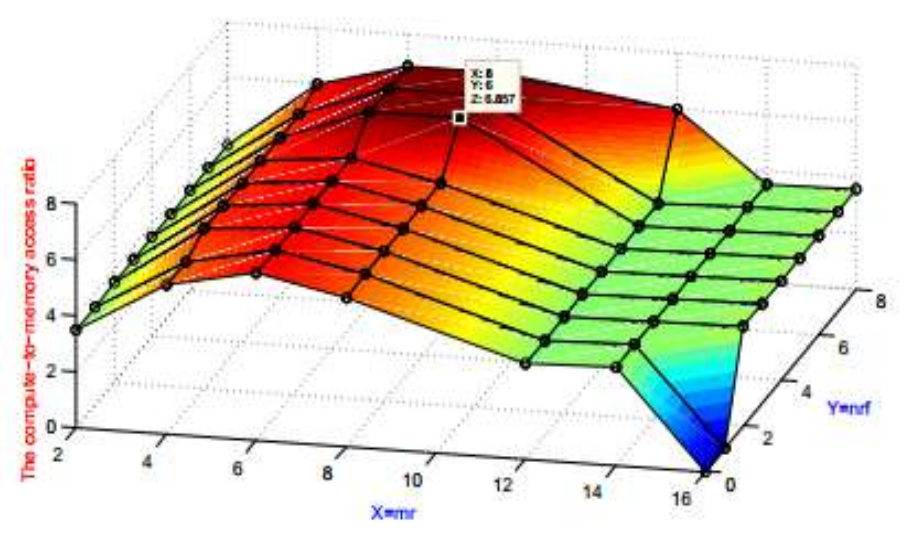
设置

nf = 32, pf = 16和element_size = 8;

nrf=6, mr×nr = 8×6;

v8-v31存储C的48个元素;

v0-v7来存储A的8个元素和B的6个元素。



寄存器内核计算-内存比率图

快速实现

寄存器分块

用一种软件实现的寄存器旋转方案，为64位ARMv8架构获得更高效的寄存器内核。

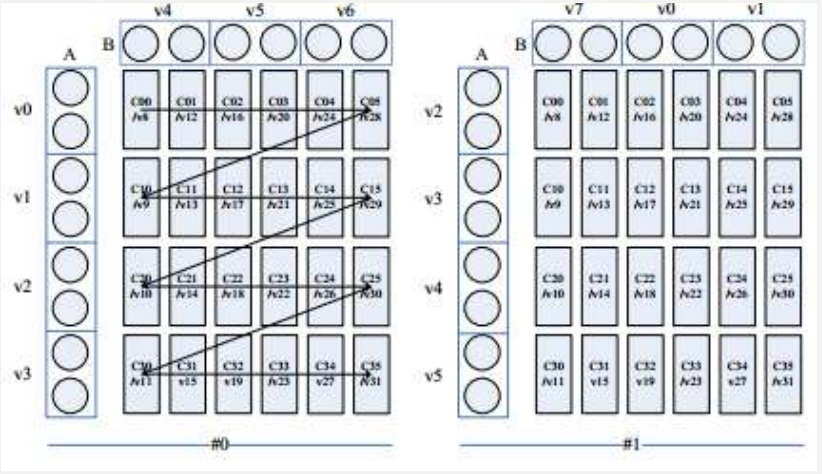
优化问题：

$$\max_{s \in S_c} \min_{v_i \in V} \; Loc('R','NF',v_i,s) - Loc('R','CL',v_i,s) \quad (12)$$

其中 $V = \{v_0, v_1, \cdots, v_7\}$ 是A和B的寄存器集，
'R'表示使用vi的读指令（fmla），
'CL'表示最后一次读取vi中的值，
'NF'表示第一次读取同一寄存器中的下一个值，
Sc是所有读指令执行顺序的集合，
Loc表示特定顺序中指令的位置。

Array	Eight Copies of the Loop Body of the Register Kernel								
	#0	#1	#2	#3	#4	#5	#6	#7	#0
A	0	2	4	7	6	1	3	5	0
	1	3	5	0	2	4	7	6	1
	2	4	7	6	1	3	5	0	2
	3	5	0	2	4	7	6	1	3
B	4	7	6	1	3	5	0	2	4
	5	0	2	4	7	6	1	3	5
	6	1	3	5	0	2	4	7	6

软件实现的寄存器旋转



8 x 6 #0和#1寄存器分配方案

最佳距离为7

快速实现

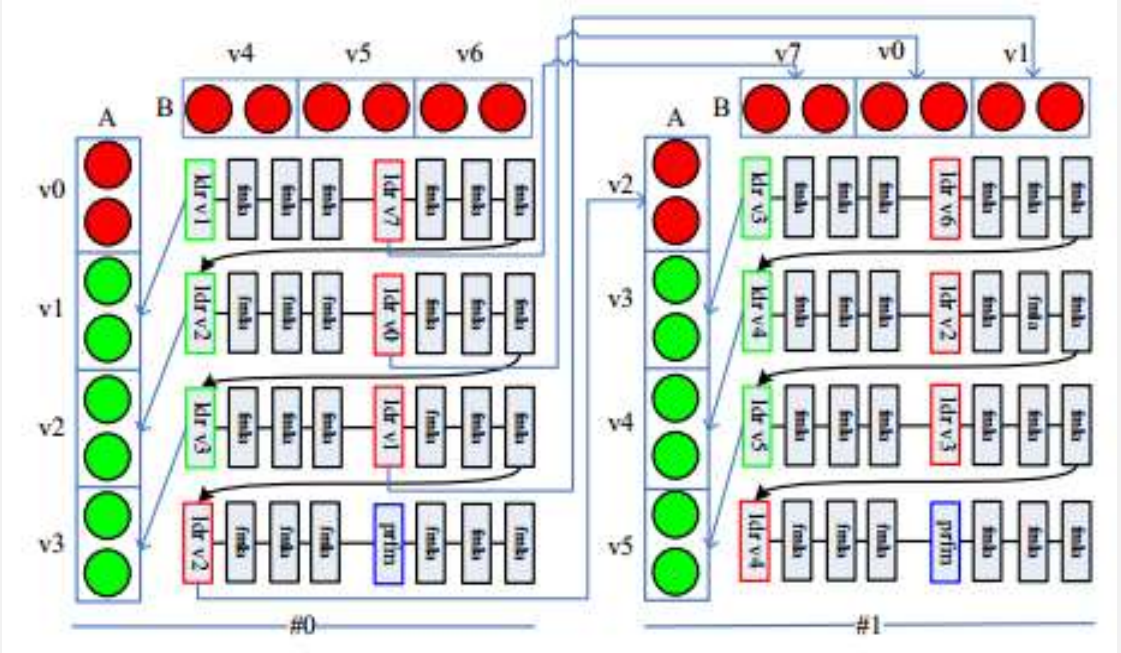
寄存器分块

- 需要考虑WAR（write-after-read）和RAW（read-after-write）依赖性；
- 必须努力隐藏加载指令的延迟，以防止流水线停顿。

优化问题：

$$\max_{s\in S}\min_{v_i\in V} \quad Loc('\mathrm{R}', v_i, s) - Loc('\mathrm{W}', v_i, s) \qquad (13)$$

其中 $V = \{\mathrm{v}_0, \mathrm{v}_1, \cdots, \mathrm{v}_7\}$ 是A和B的寄存器集，
'R'表示读指令（fmla），
'W'表示Write（ldr） 指令，
S所有可能指令执行顺序的集合，
Loc表示特定顺序中指令的位置。



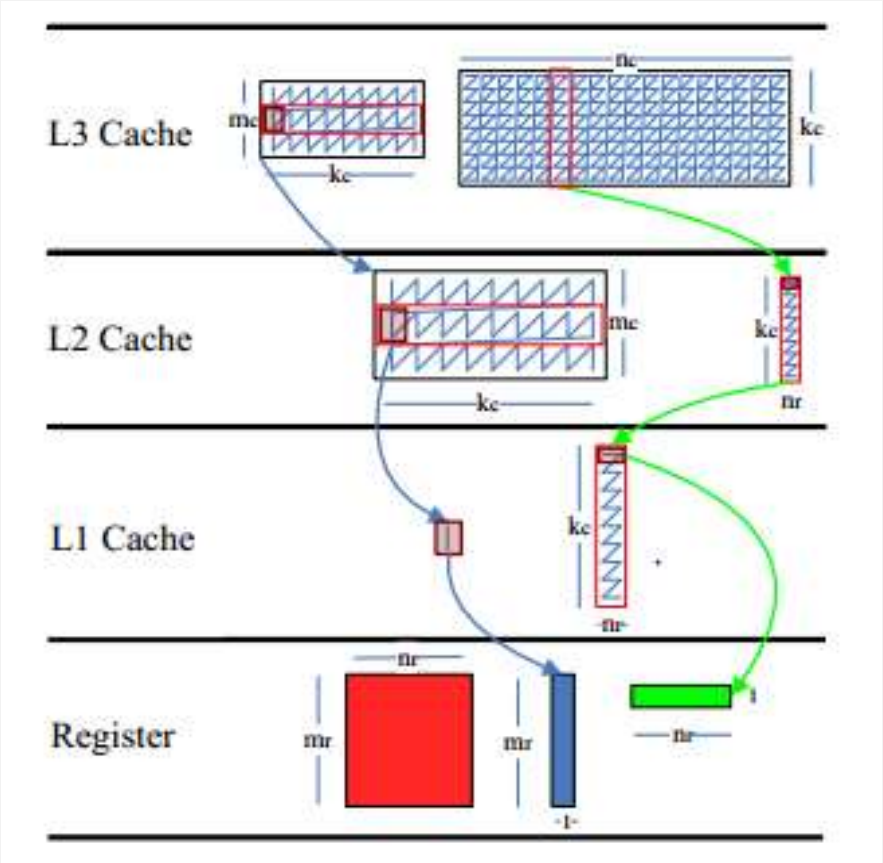
最佳RAW依赖距离指令调度

绿色寄存器在#i时加载，红色寄存器在#（i-1）%8时加载（即原始寄存器内核中较早的一次迭代）。

最佳距离为9

快速实现

局部缓存分块



- 来自B的一个kc×nr条的元素被重复使用多次，因此保留在L1缓存中；
- 来自A的mr×kc条的元素应该从L2高速缓存加载到L1高速缓存中；
- 执行2mrnrkc触发器时，需要将C子块中的mr×nr元素加载到寄存器中。

层6 GESS的计算内存访问比率：

$$\frac{2m_r n_r k_c}{(m_r k_c)_{L2\to L1} + (m_r k_c)_{L1\to R} + (k_c n_r)_{L1\to R} + (2m_r n_r)_{M\leftrightarrow R}}$$

层5 GEBS的计算到内存访问比率：

$$\frac{2m_c n_r k_c}{(m_c k_c)_{L2\to L1} + (m_c k_c)_{L1\to R} + (k_c n_r)_{L1\to R}\lceil \frac{m_c}{m_r} \rceil + (2m_c n_r)_{M\leftrightarrow R}}$$

简化的 γ：

$$\gamma = \frac{2}{\frac{2}{n_r} + \frac{1}{m_r} + \frac{2}{k_c}} \qquad (14)$$

其中mr和nr已在之前固定，所以如果kc是越大，则(14)中的比率 γ 最大化。

快速实现

局部缓存分块

考虑L1缓存的集合关联性，加两个约束：

$$k_c \times n_r \times element\_size \le \frac{(assoc1 - k1) \times L1}{assoc1}$$
$$(m_r \times n_r + m_r \times 2) \times element\_size \le \frac{k1 \times L1}{assoc1} \quad (15)$$

其中mr = 8，nr = 6，element_size = 8，L1 = 32K，assoc1 = 4（L1 cache中的路数），k1 = 1和**kc = 512**。
一个B的kc×nr条填充了L1数据缓存的3/4。

层4 GEBP的计算内存访问比率：

$$\frac{2m_c k_c n_c}{(\mathbf{m}_c\mathbf{k}_c)_{L2\to L1}\left\lceil\frac{n_c}{n_r}\right\rceil + (\mathbf{m}_c\mathbf{k}_c)_{L1\to R}\left\lceil\frac{n_c}{n_r}\right\rceil + (\mathbf{k}_c n_c)_{L1\to R}\left\lceil\frac{m_c}{m_r}\right\rceil + (\mathbf{k}_c n_c)_{L3\to L2} + (\mathbf{k}_c n_c)_{L2\to L1} + (2\,\mathbf{m}_c n_c)_{M\leftrightarrow R}}$$

简化的γ：

$$\gamma = \frac{2}{\frac{2}{\mathbf{n_r}} + \frac{1}{\mathbf{m_r}} + \frac{2}{\mathbf{k_c}} + \frac{2}{m_c}} \quad (16)$$

其中mr，nr和kc已在之前固定，所以如果mc是越大，则(16)中的比率γ最大化。

L2两个约束：

$$m_c \times k_c \times element\_size \le \frac{(assoc2 - k2) \times L2}{assoc2}$$
$$k_c \times n_r \times element\_size \le \frac{k2 \times L2}{assoc2} \quad (17)$$

其中kc = 512，nr = 6，element_size = 8，L2 = 256K，assoc2 = 16（L2cache中的路数），K2 = 2和**mc = 56**。
A的一个mc×kc块填满L2缓存的7/8。

快速实现

局部缓存分块

B的预取距离：

$$\mathrm{PRE}_B = k_c \times n_r \times element\_size = 24576.$$

A的预取距离：

$$\mathrm{PRE}_A = \alpha_{pf} num\_unroll \times m_r \times element\_size = 2 \times 8 \times 8 \times 8 = 1024$$

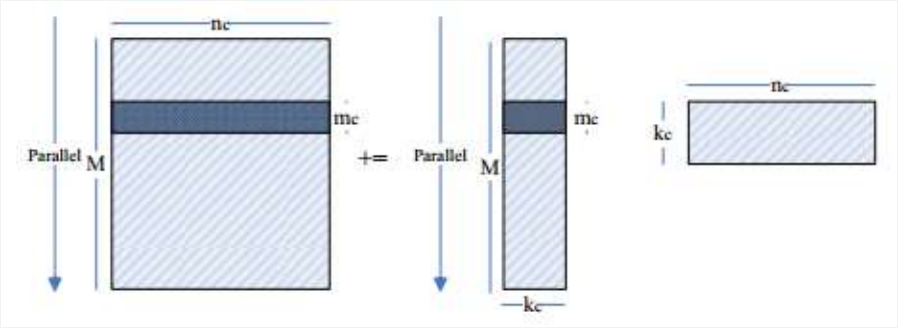
L3两个约束：

$$k_c \times n_c \times element\_size \leq \frac{(assoc3 - k3) \times L3}{assoc3}$$
$$m_c \times k_c \times element\_size \leq \frac{k3 \times L3}{assoc3} \quad (18)$$

其中mc = 56，kc = 512，L3 = 8M，assoc3 = 16（L3cache中的路数），k3 = 1和**nc = 1920。**
B的kc×nc面板（A的mc×kc 条）**填充了L3数据缓存的15/16。**

快速实现

并行缓存分块



层3循环并行

- 每个线程将被分配一个A的不同的mc×kc块；
- 所有线程共享相同B的kc×nc行面板；
- 每个线程将它自己的A块与B的共享行面板相乘。

由于一个模块中的两个核共享256KB L2缓存，因此两个线程将在同一个L2缓存中拥有自己的A块。

L2两个约束修改为：

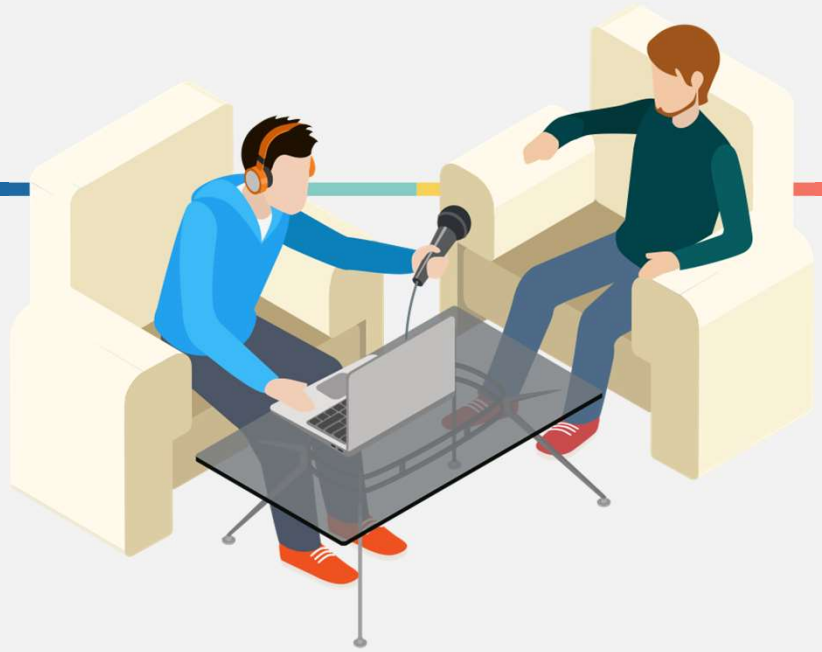
$$2 \times m_c \times k_c \times element\_size \leq \frac{(assoc2 - k2) \times L2}{assoc2}$$
$$2 \times k_c \times n_r \times element\_size \leq \frac{k2 \times L2}{assoc2} \quad (19)$$

给定nr = 6且kc = 512，得到mc = 24和k2 = 4。

多线程L3两个约束修改为：

$$k_c \times n_c \times element\_size \leq \frac{(assoc3 - k3) \times L3}{assoc3}$$
$$8 \times m_c \times k_c \times element\_size \leq \frac{k3 \times L3}{assoc3} \quad (20)$$

得到nc = 1792和k3 = 2。



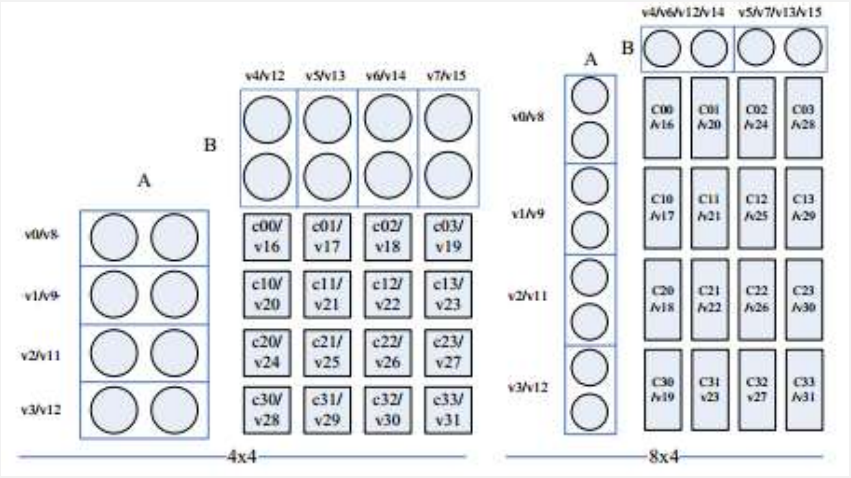
05 实验结果

- ✓ 寄存器块大小的微基准
- ✓ 性能分析

实验结果

CPU	64-bit ARMv8 eight-core processor
OS	Linux mustang-3.8.0
Compiler	gcc-4.8.2 -O2 -march=armv8-a
OpenBLAS	OpenBLAS_develop-r0.2.9
ATLAS	ATLAS 3.11.31

实验平台



寄存器内核8 x 4和4 x 4

Register Block Size	One Thread $m_r \times n_r \times k_c \times m_c \times n_c$	Eight Threads $m_r \times n_r \times k_c \times m_c \times n_c$
8×6	$8 \times 6 \times 512 \times 56 \times 1920$	$8 \times 6 \times 512 \times 24 \times 1792$
8×4	$8 \times 4 \times 768 \times 32 \times 1280$	$8 \times 4 \times 768 \times 16 \times 1192$
4×4	$4 \times 4 \times 768 \times 32 \times 1280$	$4 \times 4 \times 768 \times 16 \times 1192$

一个线程和八个线程GEBP三种块大小实现

实验结果

寄存器块大小的微基准

$LDR:FMLA$	1:1	1:2	6:16	1:3	7:24	1:4	1:5
Efficiency (%)	63.0	80.9	87.7	88.7	91.5	94.2	95.2

FMA指令的不同负载比实现的效率(理论上限)

1：2，6：16和7:24分别是LDR：FMLA比率，分别大致对应于4×4，8×4和8×6 GEBP 实现。

在每次迭代中，有（mr+nr）/2个128位内存指令将数据加载到寄存器和mrnr/2个128位NEON浮点FMA指令。

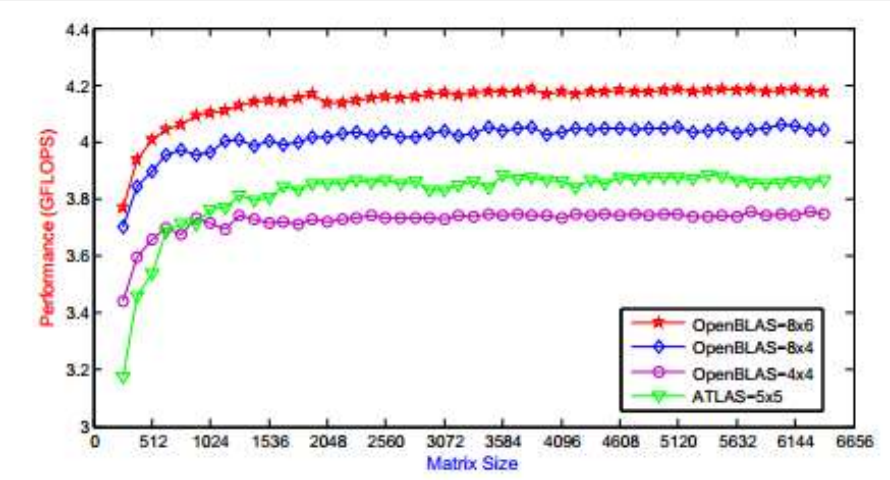
算术指令占总数的百分比为：

$$(m_r n_r/2)/(m_r n_r/2+(m_r+n_r)/2)$$

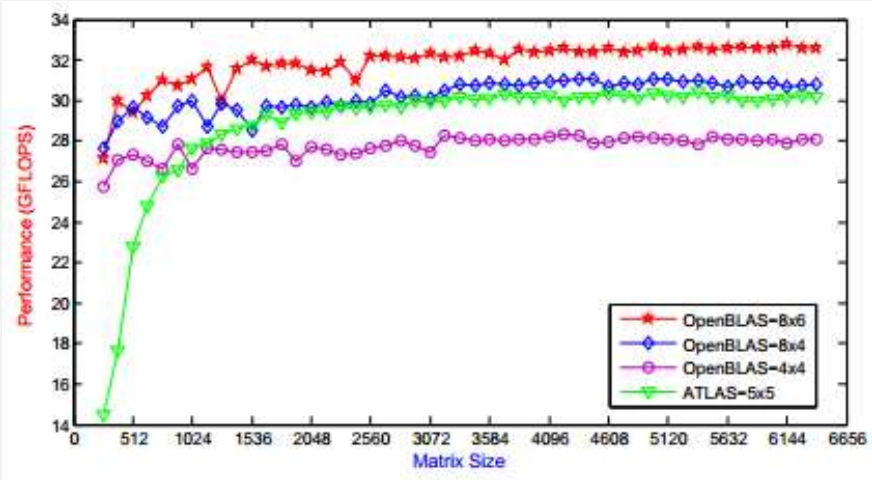
4×4，8×4和8×6 GEBP实施的百分比分别为66.7%，72.7%和77.4%。因此，8×6 GEBP将是表现最佳的。

实验结果

性能分析



四种DGEMM实现性能（一个线程）



四种DGEMM实现性能（八个线程）

- 四种DGEMM实现：
- OpenBLAS-8×6
- OpenBLAS-8×4
- OpenBLAS-4×4
- ATLAS-5×5

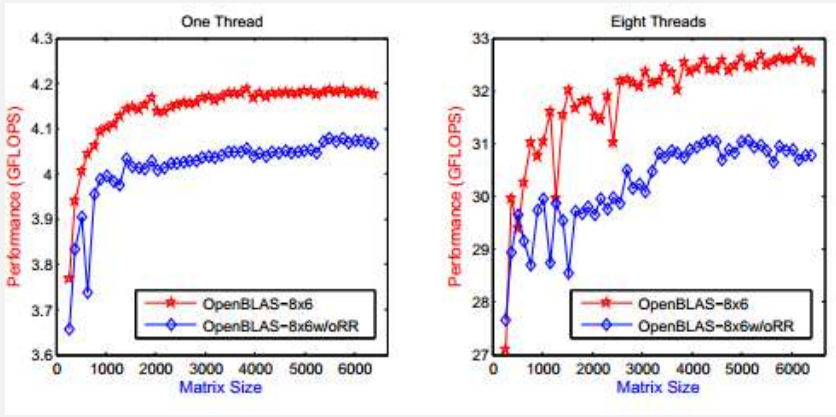
OpenBLAS-8×6在几乎所有测试的输入尺寸中都是表现最佳的 。

实验结果

性能分析

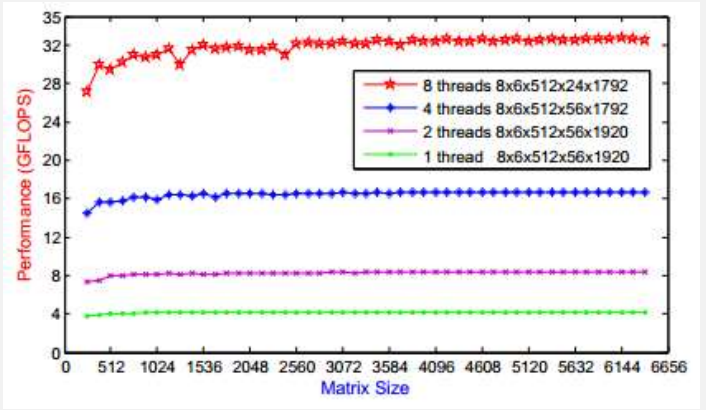
Efficiencies		OpenBLAS			ATLAS
		8 × 6	8 × 4	4 × 4	5 × 5
Peak	1 Thread	87.2%	84.6%	78.2%	80.9%
	8 Threads	85.3%	81.0%	73.7%	79.2%
Average	1 Thread	86.3%	83.6%	77.6%	79.5%
	8 Threads	83.2%	77.7%	72.3%	75.1%

四种DGEMM实现效率



软件实现的寄存器旋转效率

- 对于OpenBLAS-8×6，OpenBLAS-8×4，OpenBLAS-4×4和ATLAS-5×5，其寄存器内核的计算内存比分别为6.86 ，5.33，4和5。
- 这些结果表明我们的性能模型是合理的，计算内存访问率越大，DGEMM实现的效率就越高。
- OpenBLAS-8×6不仅高效而且可扩展。



OpenBLAS-8×6在四种线程数下的性能

实验结果

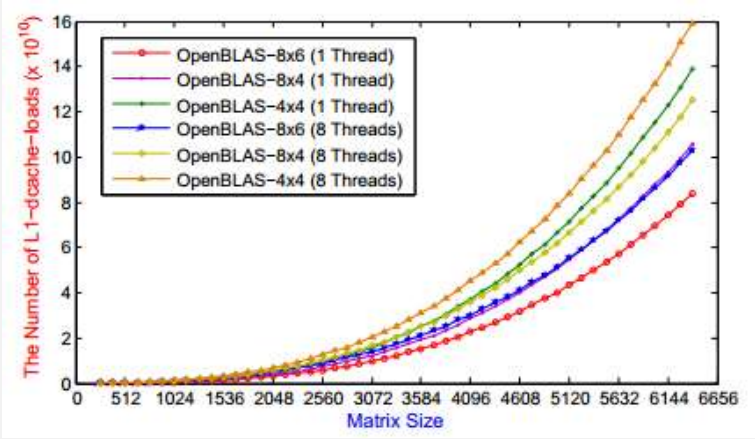
性能分析

OpenBLAS-8×6	$k_c \times m_c \times n_c$	Peak efficiency (%)	Average efficiency (%)
Serial	**512 × 56 × 1920**	87.2	86.3
	320 × 96 × 1536	86.4	85.4
Parallel (8 Threads)	**512 × 24 × 1792**	85.3	83.2
	512 × 24 × 1920	85.2	82.9
	512 × 56 × 1792	80.4	75.5
	512 × 56 × 1920	80.1	75.4

OpenBLAS-8×6在不同块大小下性能

	OpenBLAS-8×6	OpenBLAS-8×4	OpenBLAS-4×4
One Thread	5.2%	4.3%	5.7%
Eight Threads	3.6%	3.2%	5.0%

OpenBLAS-8×6，OpenBLAS-8×4，OpenBLAS-4×4 L1 cache未命中率



OpenBLAS-8×6，OpenBLAS-8×4，OpenBLAS-4×4 L1-dcache负载数

- 在串行和并行设置中，OpenBLAS-8×6具有最小的L1-dcache-负载数。
- 在任何一种情况下， OpenBLAS-8×6都不会产生最低的L1 cache未命中率。

结　论

- 介绍了一种64位ARMv8多核处理器的高效DGEMM的设计和实现方法，构建了性能模型。
- 实现的方法达到了峰值性能，非常接近微基准测试建立的理论上限。
- 并行实现在评估的各种不同矩阵大小上实现了良好的可扩展性。

谢　　谢！

THESIS DEFENSE POWERPOINT TEMPLATE

汇报人：刘宜佳、郭凌超、王东紫