ARM嵌入式系统的DNN性能优化

张先轶
PerfXLab澎峰科技
xianyi@perfxlab.com

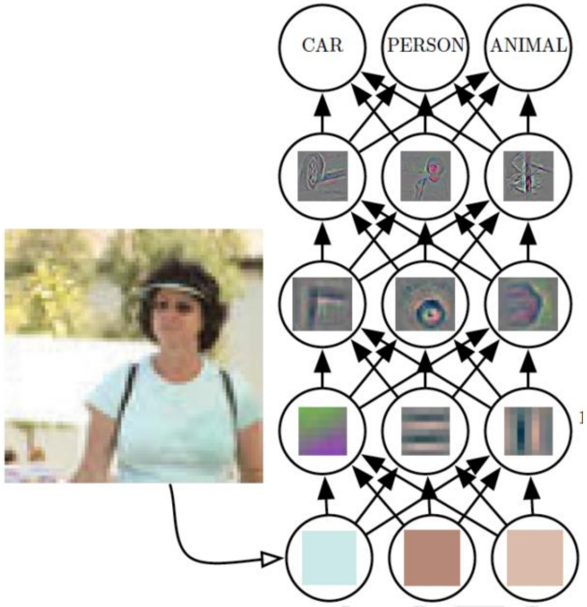
PerfXLab澎峰科技

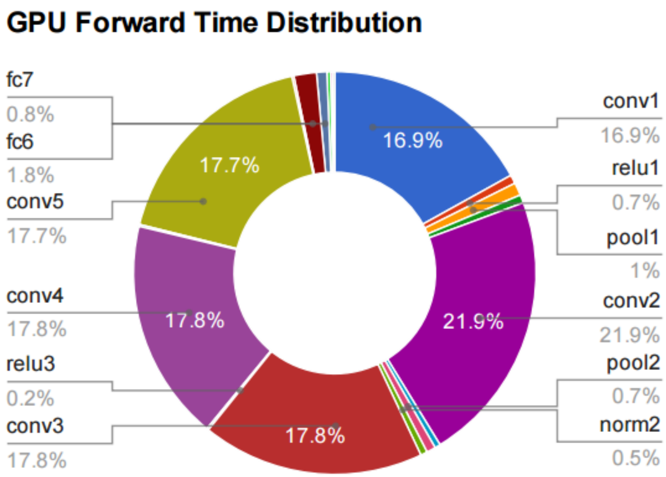
- AI+计算
- 深度学习
  - 服务器 + 嵌入式终端
  - 框架：PerfNet （基于mxnet）
  - 性能库：PerfDNN
    - x86, ARM, POWER
    - 支持低精度
- PerfCV
  - 基本CV类功能（cvt_color, resize...）
- OpenBLAS

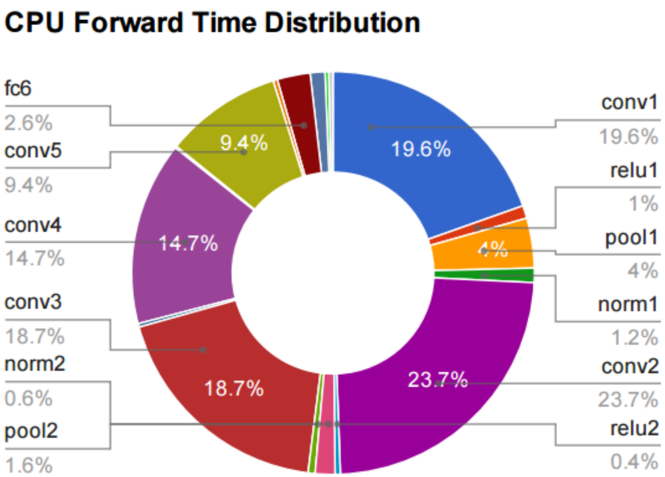
PerfXLab

Deep Learning

- 大数据+大计算
- Alexnet
    - Conv layer ->BLAS
    - FC layer -> BLAS

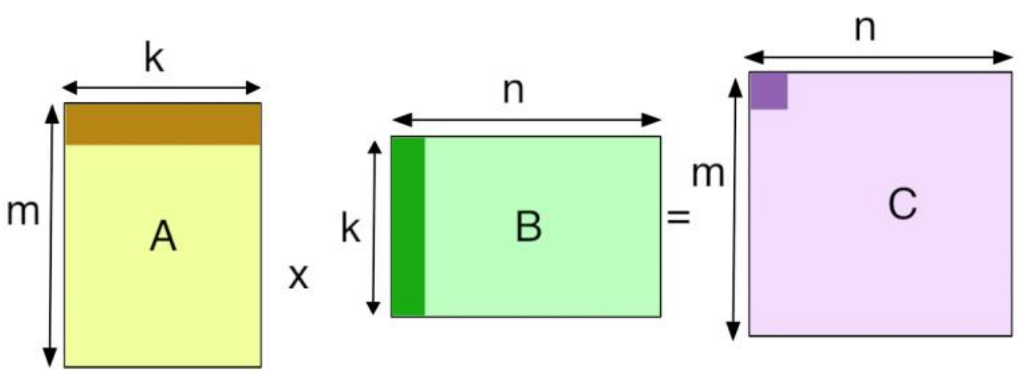


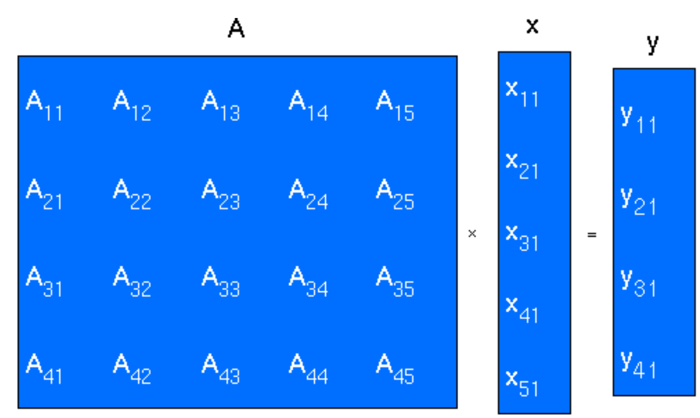




什么是BLAS？

- Basic Linear Algebra Subprograms
- 基本线性代数子程序
  - BLAS3级：矩阵-矩阵
  - BLAS2级：矩阵-向量
  - BLAS1级：向量-向量





OpenBLAS

- 2011年， forked from GotoBLAS2
- 全球最好的开源矩阵计算库
- 2016 中国计算机学会科技进步二等奖
- 进入主流Linux发行版
- 进入OpenHPC套件

Ubuntu
openblas package

fedora

gentoo linux™

debian GNU/Linux

OpenBLAS

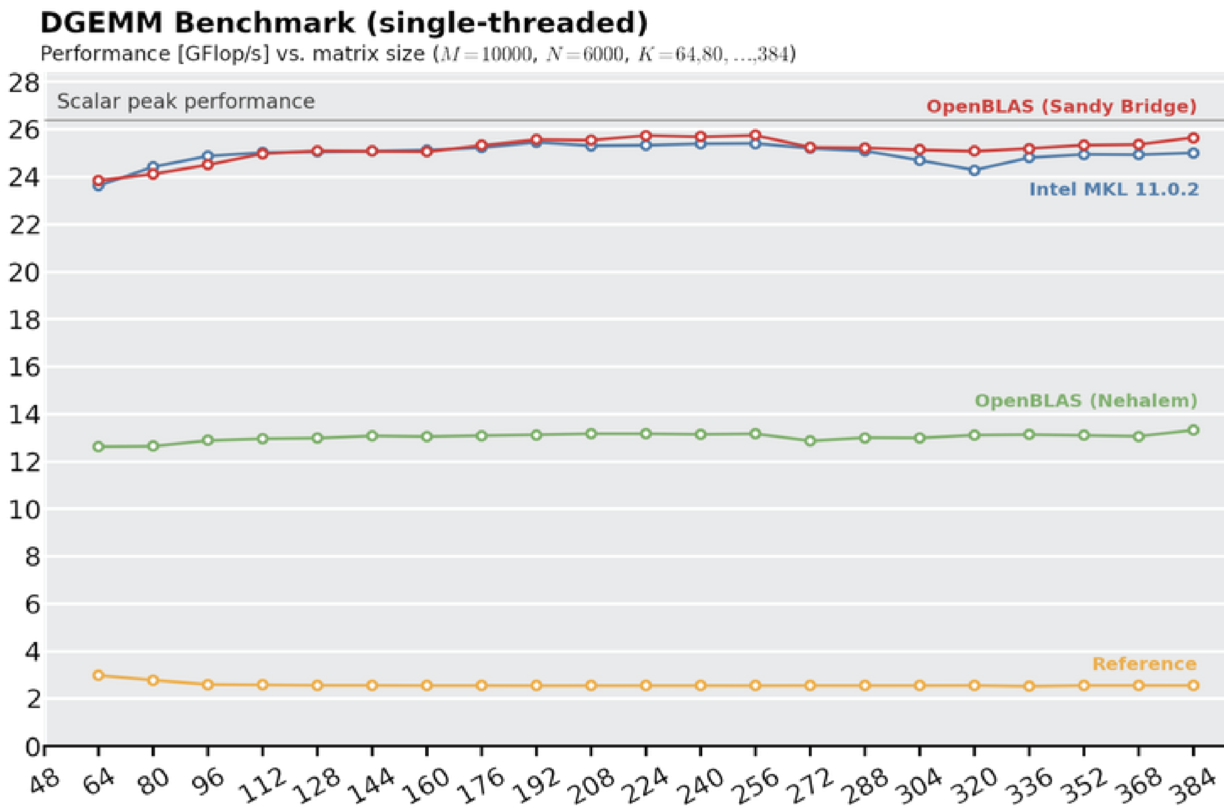
- 支持主流CPU处理器
  - Intel, AMD
  - ARM, AArch64
  - MIPS, 龙芯
  - IBM POWER
- 支持常见操作系统
  - Linux
  - Windows
  - Mac OSX
  - FreeBSD
  - Android

OpenBLAS用户

julia
GNU Octave
dmlc mxnet
Caffe
IBM
ARM
NVIDIA
Ceemple
VERSATUShpc HIGH PERFORMANCE CONSULTING
PathScale High Performance Compilers
西安電子科技大學 XIDIAN UNIVERSITY 1931
Sogou搜狗
華僑大學 HUAQIAO UNIVERSITY

OpenBLAS性能

- Intel Sandy Bridge



OpenBLAS性能

- 龙芯3A

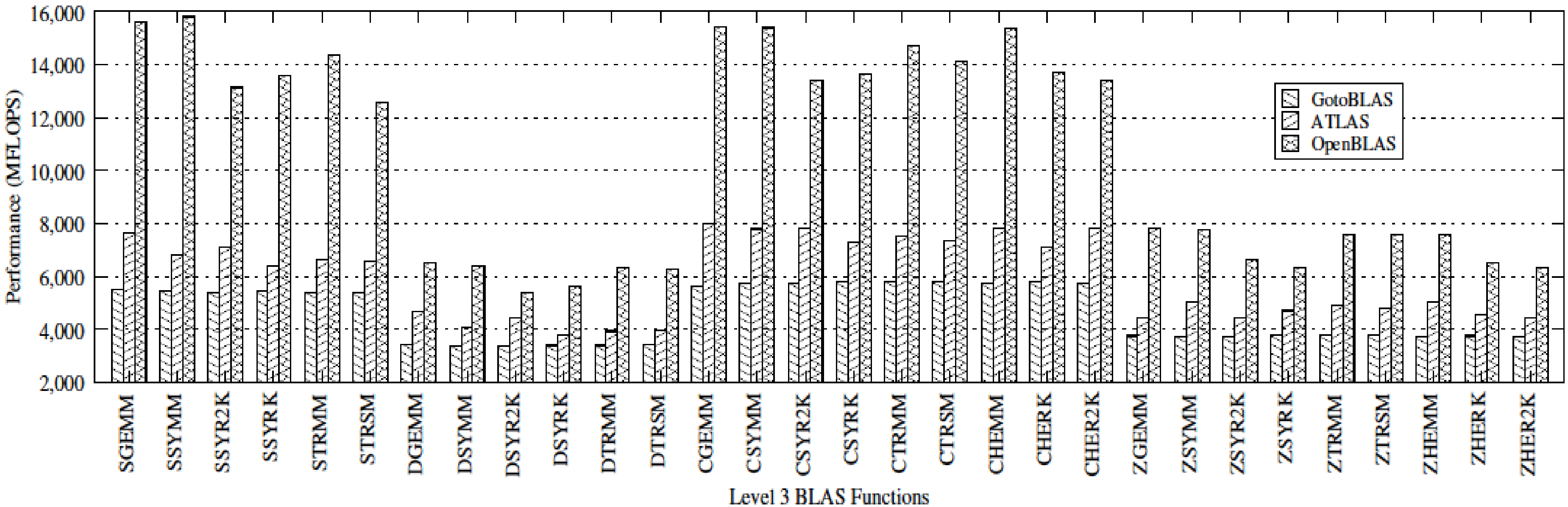
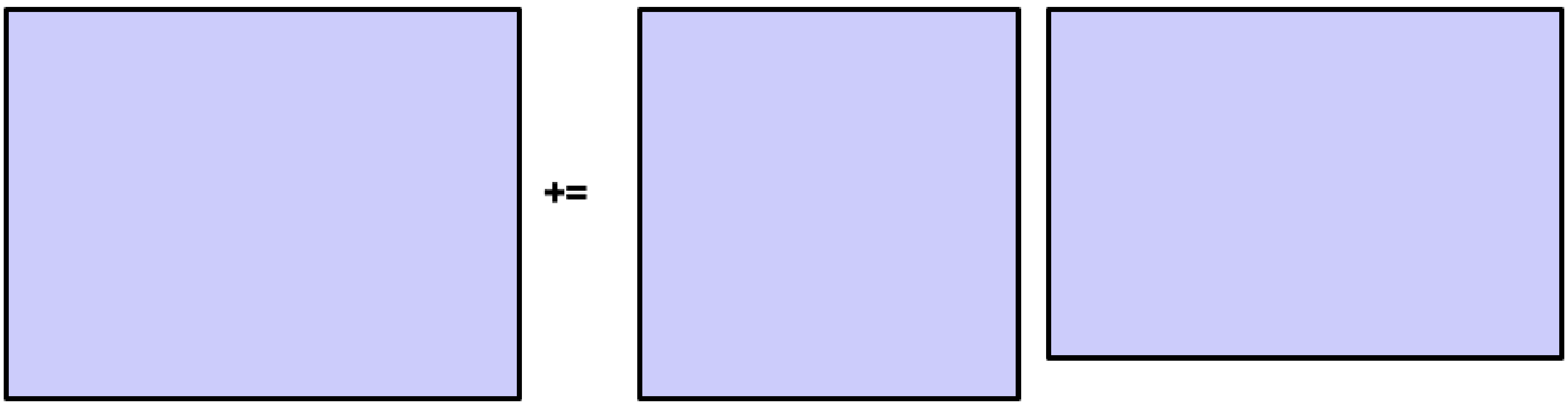


Figure 11. Multi-threaded Level 3 BLAS Performance (NP=4)

GEMM矩阵乘法

C:MxN　　A:MxK　　B: KxN

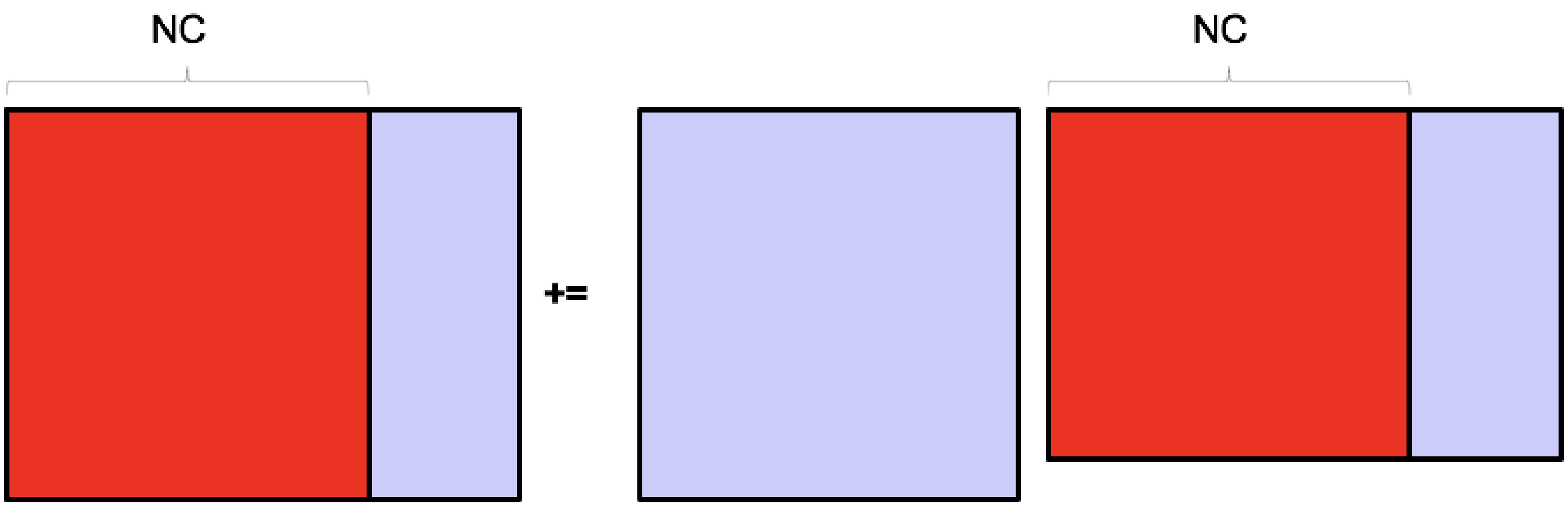


- 简单实现：ijk三重循环
- 分块
  - 提高Cache利用率
  - 怎么分块？

```
for(i=0; i<M; i++)
  for(j=0; j<N, j++)
    for(k=0; k<K; k++)
      C[i][j]+=A[i][k]*B[k][j]
```

GEMM矩阵乘法

- N方向分块

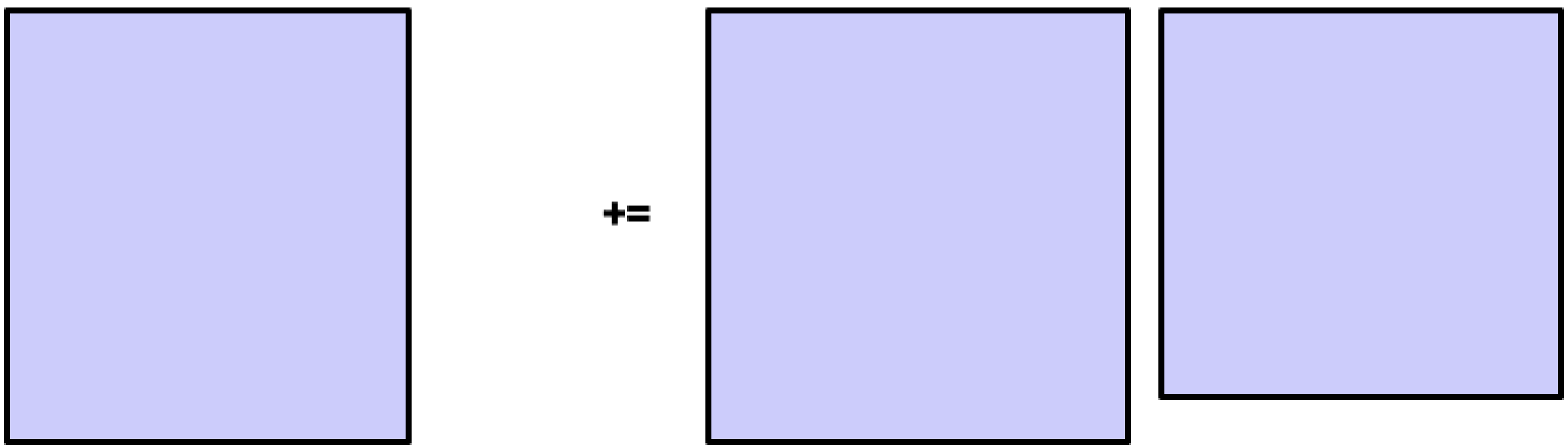


GEMM矩阵乘法

C： MxNc

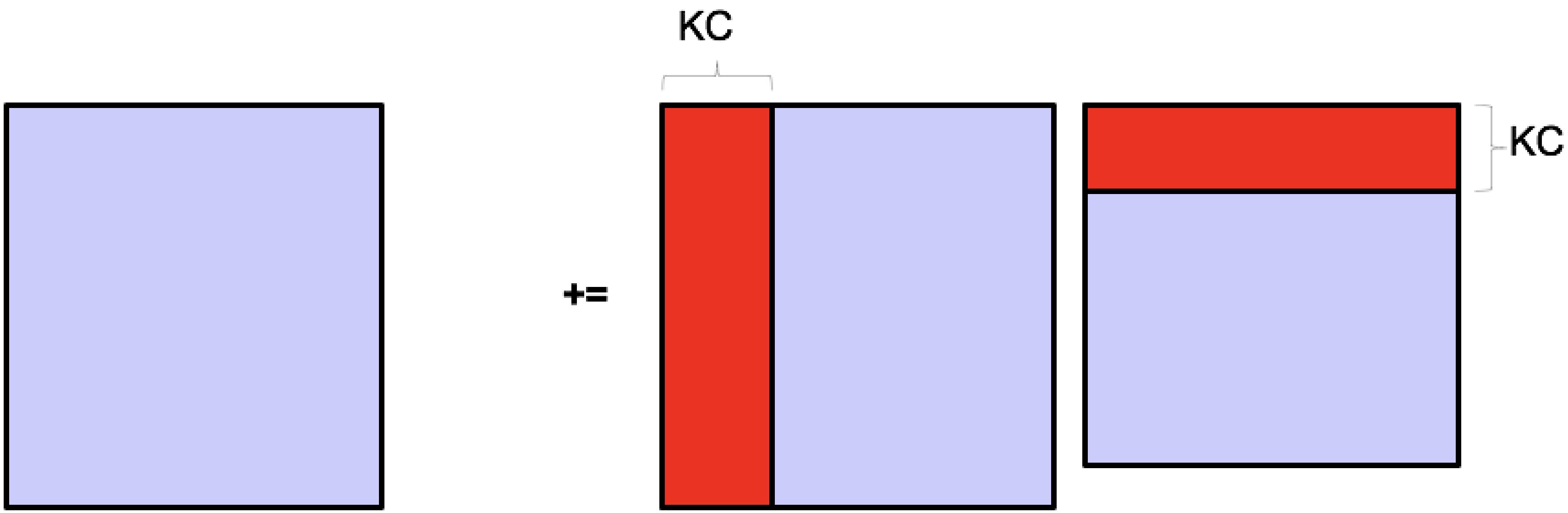
A: MxK

B： KxNc



GEMM矩阵乘法

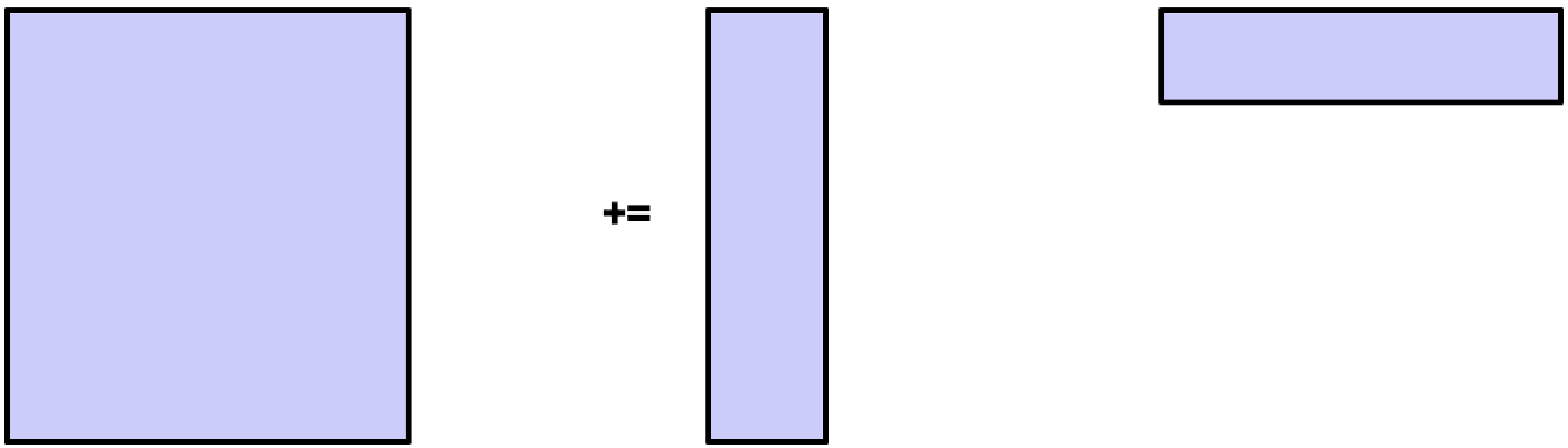
K方向分块



GEMM矩阵乘法

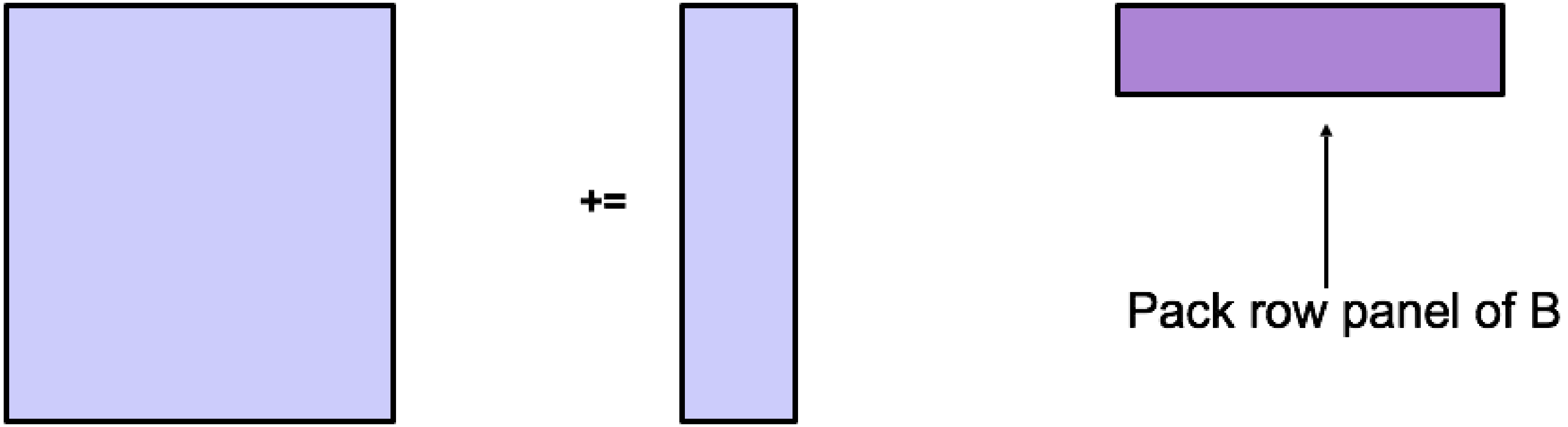
GEPP

C： MxNc A: MxKc B： Kc x Nc



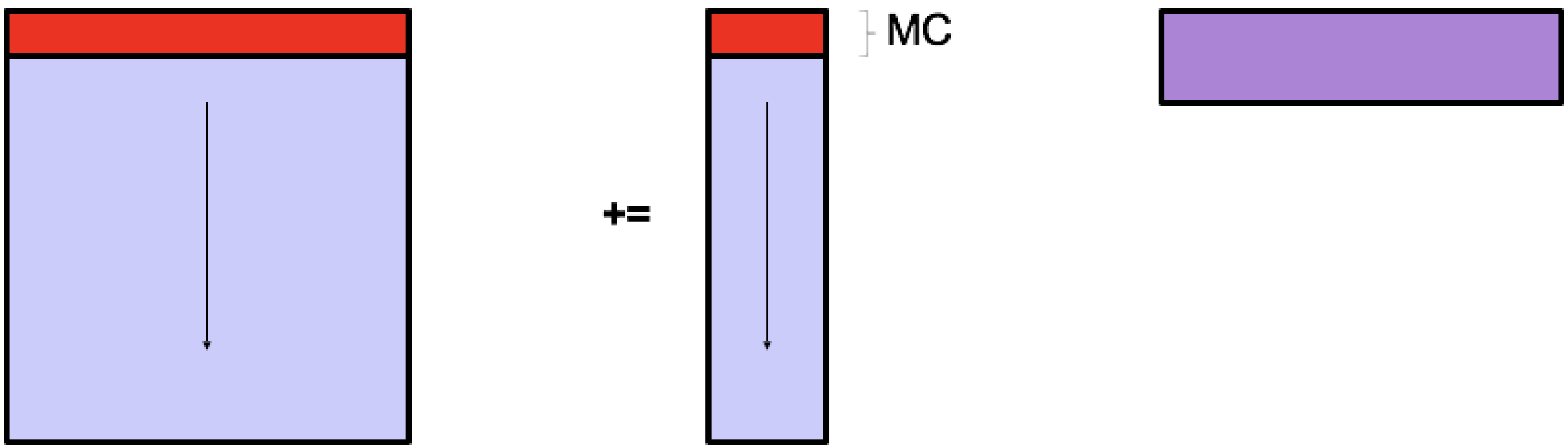
GEMM矩阵乘法

- B部分打包，连续存储
  - 降低cache miss



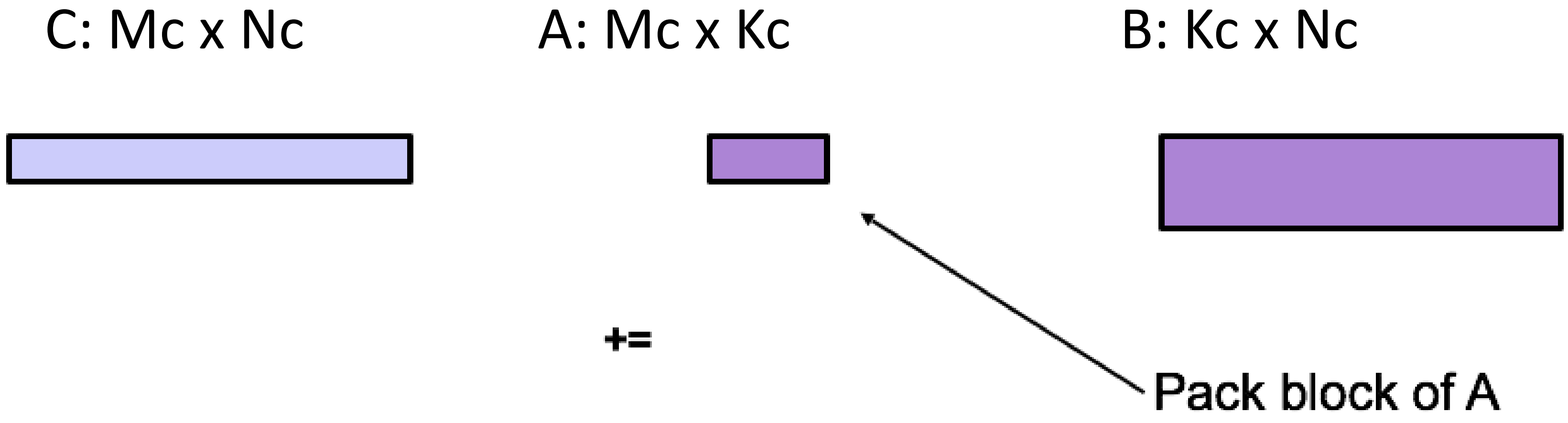
GEMM矩阵乘法

- M方向分块



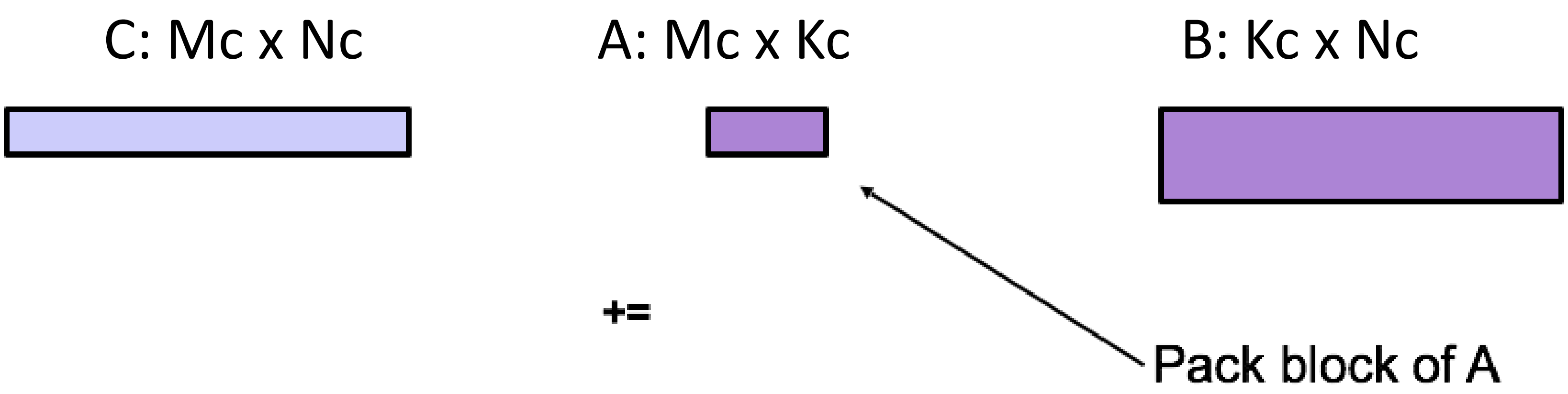
GEMM矩阵乘法

- GEBP
- A打包



GEMM矩阵乘法

- 核心汇编优化
  - 寄存器分块
  - SIMD指令
  - 指令流水线优化，循环展开，重排，预取



BLAS性能优化流派

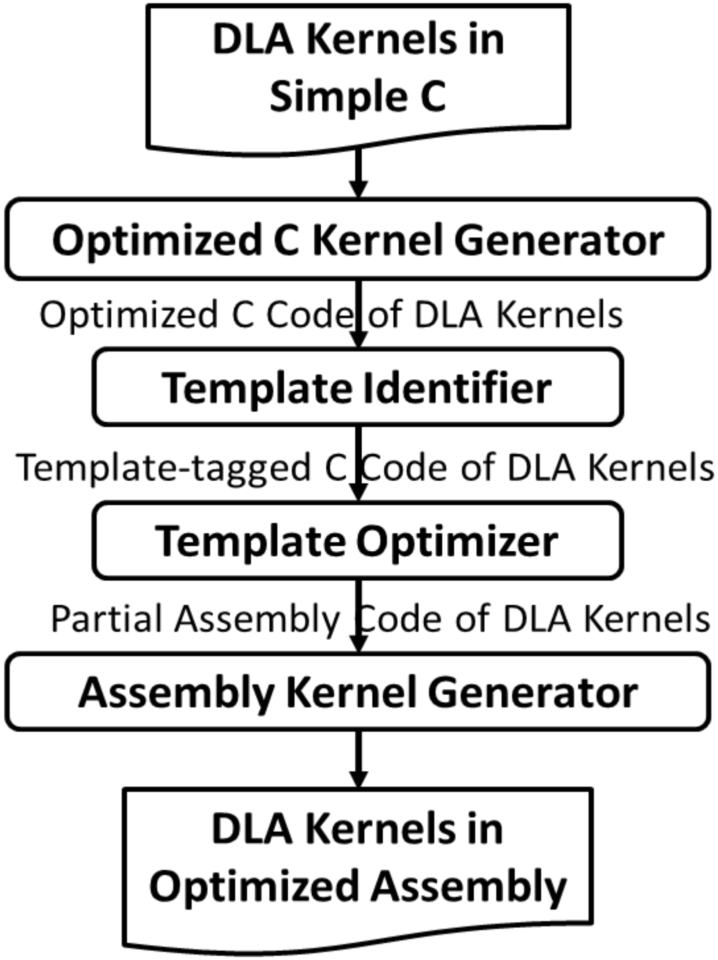
- 自动调优Auto-tuning
  - ATLAS
  - 快速开发和移植
  - 性能？？
- 手工核心汇编
  - GotoBLAS/OpenBLAS
  - 性能好
  - 新架构？
- Auto-tuning生成高效汇编代码？？

AUGEM

- Automatically Generate Efficient Matrix kernel
- 目标：自动生成BLAS中高效汇编
- 支持x86 ISA
  - SSE，AVX，AVX 2.0
- 支持ARMv7 ISA
  - Neon

AUGEM

- 基于Template
  - 隐含手工优化知识
- 输入
  - BLAS Kernel功能描述
- 输出
  - 高性能汇编



C级别Kernel优化

Input simple C code of gemm kernel

```
void gemmkernel(int  Mc, int  Nc,  int  Kc,  double  alpha, \
    double  *A, double *B, double  *C, int  LDC) {
    int i, j, l;
    double res, tmp1, tmp2, tmp3;
    for (j = 0; j < Nc; j += 1) {
        for (i = 0; i < Mc; i += 1) {
            res = 0;
            for (l = 0; l < Kc; l += 1) {
                tmp1 = A[l * Mc + i];
                tmp2 = B[j * Kc + l];
                tmp3 = tmp1 * tmp2;
                res = res + tmp3;
            }
            res = res * alpha;
            tmp1 = C[j * LDC + i];
            res = res + tmp1;
            C[j * LDC + i] = res;
}}}
```

Output Optimized C code of gemm kernel

```
void gemmkernel (int Mc, int Nc, int Kc, double alpha, \
     double *A, double *B, double *C, int LDC) {
     int i, j, l,...variables declaration;
     for (j = 0; j < Nc;  j += 2) {
         ptr_A = A; ptr_C0 = C; ptr_C1 = C + LDC;
         pre_B = B + 2 * Kc;
         for (i = 0; I < MC; I += 2) {
             Prefetch(pre_B); ptr_B = B;
             res0 = 0; res1 = 0; res2 = 0; res3 = 0;
             for (l = 0; l < Kc; l += 4) {
                 Prefetch(ptr_A+PREDIS);
                 tmp0 = ptr_A[0];
                 tmp1 = ptr_B[0];
                 tmp2 = tmp0 * tmp1;
                 res0 = res0 + tmp2;
                 tmp0 = ptr_A[1];
                 tmp1 = ptr_B[0];
                 tmp2 = tmp0 * tmp1;
                 res1 = res1 + tmp2;
                 ...
                 ptr_A += 4; ptr_B += 4;
             } ...cleanup code for loop l
             res0 = res0 * alpha; res1 = res1 * alpha; ...
             tmp0 = C[j * LDC +i];
             res0 = res0 + tmp0;
             C[j * LDC + i] = res0;
             ...
         }...cleanup code for loop i
}}
```

Template识别

Output Optimized C code of gemm kernel

```
void gemmkernel (int Mc, int Nc, int Kc, double alpha, \
    double *A, double *B, double *C, int LDC) {
    int i, j, l,...variables declaration;
    for (j = 0; j < Nc;  j += 2) {
        ptr_A = A; ptr_C0 = C; ptr_C1 = C + LDC;
        pre_B = B + 2 * Kc;
        for (i = 0; I < MC; I += 2) {
            Prefetch(pre_B); ptr_B = B;
            res0 = 0; res1 = 0; res2 = 0; res3 = 0;
            for (l = 0; l < Kc; l += 4) {
                Prefetch(ptr_A+PREDIS);
                tmp0 = ptr_A[0];
                tmp1 = ptr_B[0];
                tmp2 = tmp0 * tmp1;
                res0 = res0 + tmp2;
                tmp0 = ptr_A[1];
                tmp1 = ptr_B[0];
                tmp2 = tmp0 * tmp1;
                res1 = res1 + tmp2;
                ...
                ptr_A += 4; ptr_B += 4;
            } ...cleanup code for loop l
            res0 = res0 * alpha; res1 = res1 * alpha; ...
            tmp0 = C[j * LDC +i];
            res0 = res0 + tmp0;
            C[j * LDC + i] = res0;
            ...
        }...cleanup code for loop i
}}
```

```
Load:      tmp0 = ptr_A[0];
Load:      tmp1 = ptr_B[0];
Multiply:  tmp2 = tmp0 * tmp1;
Add:       res0 = res0 + tmp2;
```

```
Load:  tmp0 = C[j*LDC+i];
Add:   res0 = res0 + tmp0;
Store: C[j*LDC+i] = res0;
```

Template识别

- 预定义6种
  - 分为两组

Atomic Templates	Compound Templates
mmCOMP(A,idxa,B,idxb,res) 1. tmp0 = A[idxa]; 2. tmp1= B[idxb]; 3. tmp2 = tmp0 * tmp1; 4. res = res + tmp2;	**mmUnrolledCOMP**(A,idxa,na,B,idxb,nb,res) 1. mmCOMP(A,idxa,B,idxb,res_0) 2. mmCOMP(A,idxa+1,B,idxb,res_1) 3. mmCOMP(A,idxa+na-1,B,idxb,res_{na-1}) 4. ... 5. mmCOMP(A,idxa+na-1,B,idxb+nb-1,$res_{(na-1)x(nb-1)}$)
mmSTORE(A,idx,res) 1. tmp0 = A[idx] 2. res = res + tmp0; 3. A[idx] = res;	**mmUnrolledSTORE**(A,idx,n,res) 1. mmSTORE(A,idx,res_0) 2. ... 3. mmSTORE(A,idx+n-1,res_{n-1})
mvCOMP(A,idxa,B,idxb,scal) 1. tmp0 = A[idxa]; 2. tmp1 = B[idxb]; 3. tmp0 = tmp0 * scal; 4. tmp2 =tmp1 + tmp0; 5. B[idxb] = tmp2;	**mvUnrolledCOMP**(A,idxa,B,idxb,n,scal) 1. mvCOMP(A,idxa,B,idxb,scal) 2. mvCOMP(A,idxa+1,B,idxb+a,scal) 3. ... 4. mvCOMP(A,idxa+n-1,B,idxb+n-1,scal)

Template优化

- SIMD向量化

```
mmUnrolledCOMP(ptr_A,0,2,ptr_B,0,2,(res0,r
es1,res2,res3))
    mmCOMP(ptr_A,0,ptr_B,0,res0)
        1.tmp0 = ptr_A[0];
        2.tmp1 = ptr_B[0];
        3.tmp2 = tmp0 * tmp1;
        4.res0 = res0 + tmp2;
    mmCOMP(ptr_A,1,ptr_B,0,res1)
        1.tmp0 = ptr_A[1];
        2.tmp1 = ptr_B[0];
        3.tmp2 = tmp1 * tmp1;
        4.res1 = res1 + tmp2;
    mmCOMP(ptr_A,0,ptr_B,1,res2)
        1.tmp0 = ptr_A[0];
        2.tmp1 = ptr_B[1];
        3.tmp2 = tmp0 * tmp1;
        4.res2 = res2 + tmp2;
    mmCOMP(ptr_A,1,ptr_B,1,res3)
        1.tmp0 = ptr_A[1];
        2.tmp1 = ptr_B[1];
        3.tmp2 = tmp0 * tmp1;
        4.res3 = res3 + tmp2;
```

1. Vld ptr_A, 0, vec0
2. Vdup ptr_B, 0, vec1
3. Vmul vec0, vec1, vec2
4. Vadd vec2, vec3, vec3

1. Vld ptr_A, 0, vec4
2. Vdup ptr_B, 1, vec5
3. Vmul vec4, vec5, vec6
4. Vadd vec6, vec7, vec7

Template优化

- 寄存器分配
  - 根据用途分组
    - A (vec0, vec4)
    - B (vec1, vec5)
    - C (vec3, vec7)
    - 中间结果 (vec2, vec6)
  - 不考虑寄存器溢出
    - 临时保存到堆栈
    - 影响性能

1. Vld ptr_A, 0, vec0
2. Vdup ptr_B, 0, vec1
3. Vmul vec0, vec1, vec2
4. Vadd vec2, vec3, vec3
5. Vld ptr_A, 0, vec4
6. Vdup ptr_B, 1, vec5
7. Vmul vec4, vec5, vec6
8. Vadd vec6, vec7, vec7

汇编指令映射

1. Vld ptr_A, 0, reg0
2. Vdup ptr_B, 0, reg1
3. Vmul reg0, reg1, reg2
4. Vadd reg2, reg3, reg3
5. Vld ptr_A, 0, reg4
6. Vdup ptr_B, 1, reg5
7. Vmul reg4, reg5, reg6
8. Vadd reg6, reg7, reg7

Instructions	SSE	AVX
Vld array, offset, reg	Vld offset(array),reg	Vld offset(array),reg
Vst reg, array, offset	Vst offset(array),reg	Vst offset(array),reg
Vmul reg0,reg1,reg2 Vadd reg2,reg3,reg3	Vmov reg1,reg2 Vmul reg0,reg1 Vadd reg1,reg3	Vmul reg0,reg1,reg2 Vadd reg2,reg3,reg3
…	…	…

1. Vld 0(ptr_A), reg0
2. Vdup 0(ptr_B), reg1
3. Vmov reg1,reg2
4. Vmul reg0, reg1
5. Vadd reg1, reg3
6. …

1. Vld 0(ptr_A), reg0
2. Vdup 0(ptr_B), reg1
3. Vmul reg0, reg1, reg2
4. Vadd reg2, reg3, reg3
5. …

汇编生成

- 将剩余代码转换成汇编
  - 循环控制
- 保持寄存器分配一致性
  - 引入reg_table
  - 全局记录表

Algorithm of Template Optimizer

Input: *input*: template-annotated kernel in low-level C
arch: architecture specification
Output: *res*: optimized kernel in assembly

$res = input$;
$reg\_table =$ empty;
reg_free=available_registers($arch$);
for each annotated code region r in $input$ **do**
  $r\_annot =$ template_annotation(r);
  $r1 =$ Optimizer[r_annot]
  (r, reg_table, reg_free, $arch$);
  $res =$ replace $r1$ with r in res;
end for

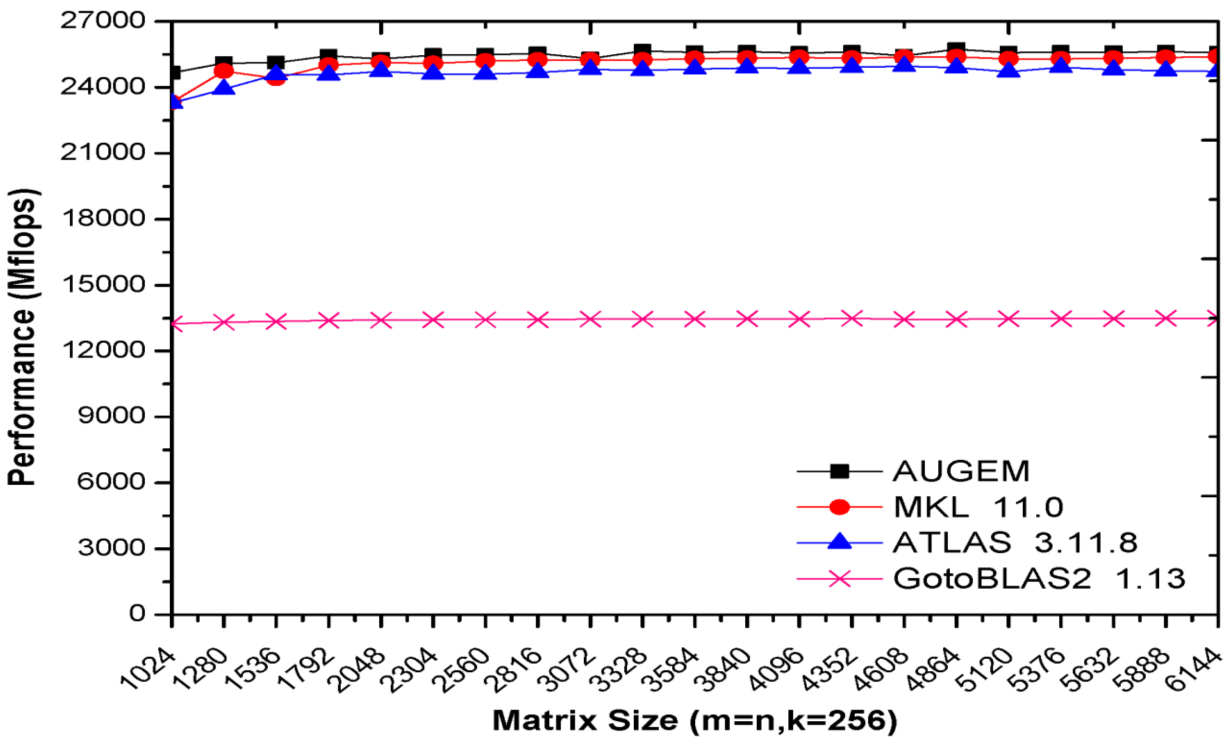
AUGEM性能测试

- X86平台

CPU	Intel Sandy Bridge 8C E5-2680 (2.7GHz)	AMD Piledriver 6380 Processor (2.5GHz)
L1d Cache	32KB	16KB
L2 Cache	256KB	2048KB
Vector Size	256-bit	256-bit
Core(s) per socket:	8	8
CPU socket(s)	2	2
Compiler	gcc-4.7.2	SAME
GotoBLAS	GotoBLAS2 1.13 BSD version	SAME
ATLAS	ATLAS 3.11.8 version	SAME
MKL	MKL 11.0 updated 2	N/A
ACML	N/A	ACML 5.3.0 version

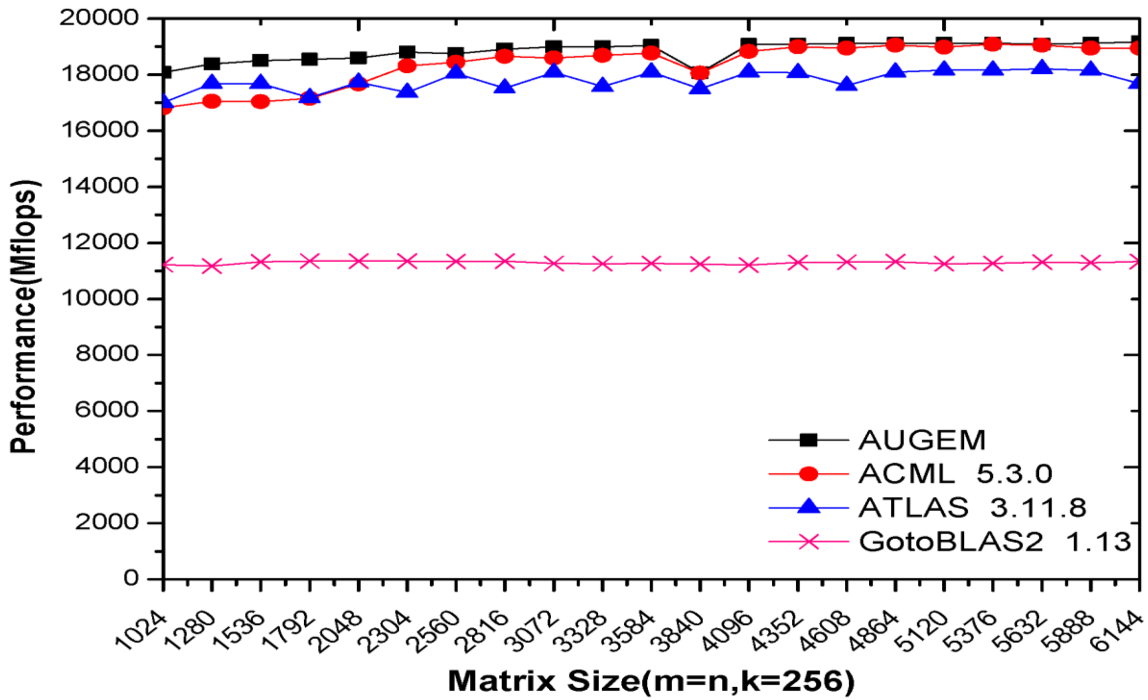
AUGEM性能测试

- DGEMM on Intel Sandy Bridge



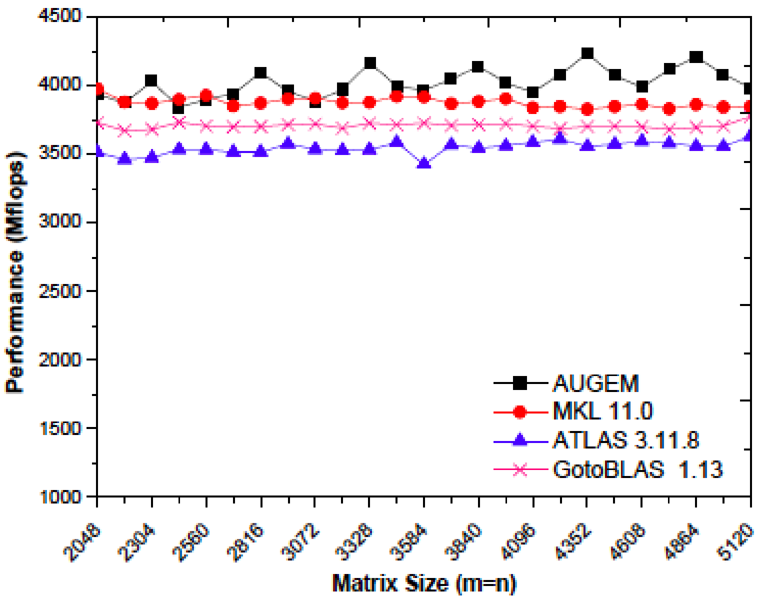
AUGEM性能测试

- DGEMM on AMD Piledriver

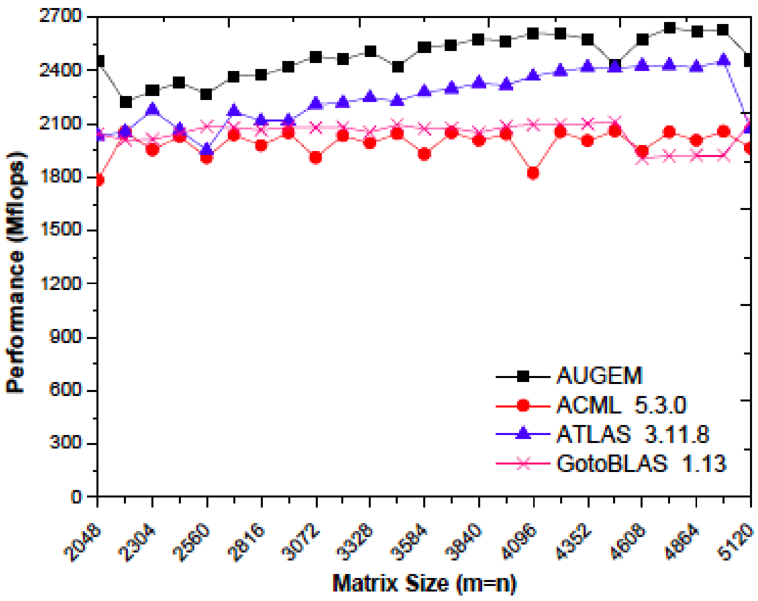


AUGEM性能测试

- DGEMV (BLAS 2级)



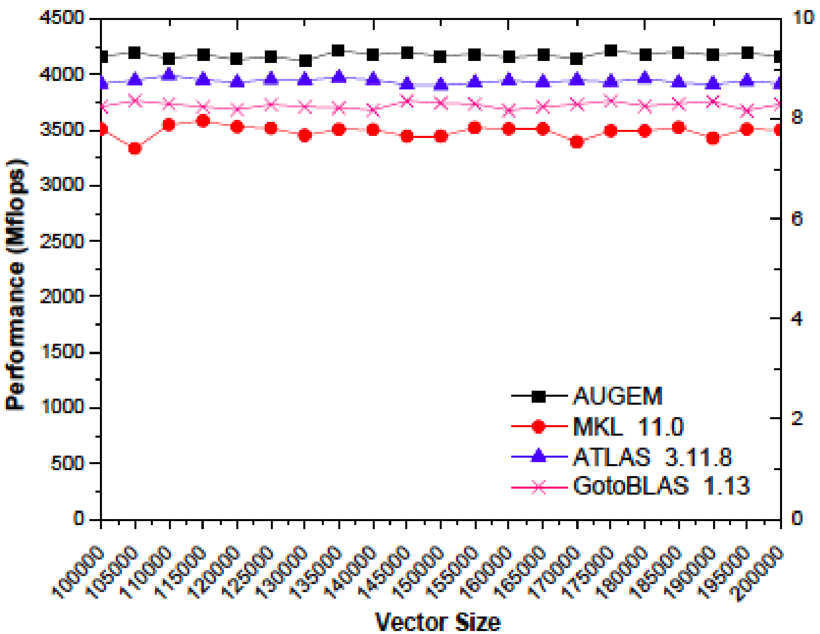
(a) SandyBridge



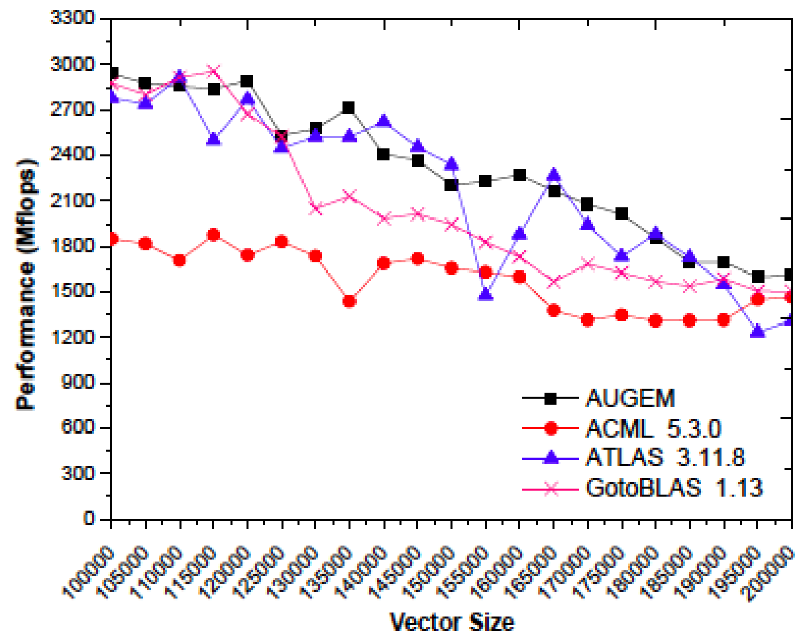
(b) Piledriver

AUGEM性能测试

- DAXPY (BLAS 1级)



(a) SandyBridge



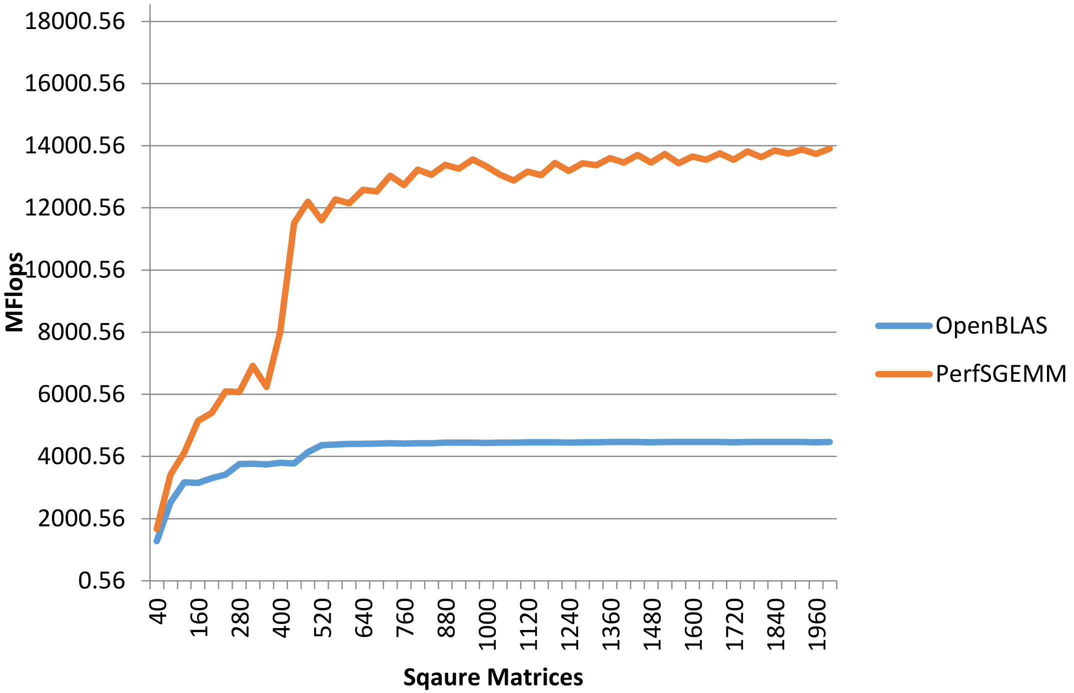
(b) Piledriver

SGEMM on ARMv7

- OpenBLAS没有用向量Neon指令
- Neon SIMD指令
    - 与IEEE 754标准不一致
    - Round mode
    - 不影响深度学习应用的精度
- PerfDNN (PerfSGEMM)
    - 面向深度学习
    - conv, fc
- AUGEM增加ARM指令
    - 支持ARM特殊指令
        - vmla.f32 c, a, b[d]

PerfSGEMM on ARMv7

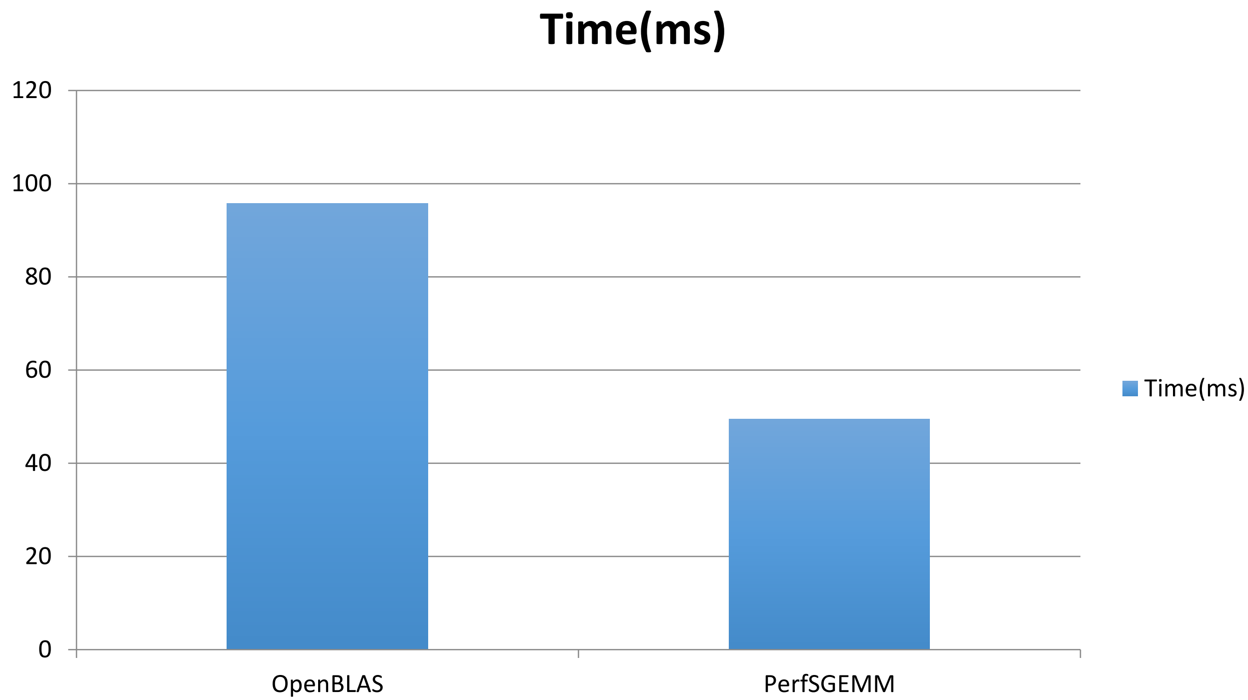
- ARM Cortex A15 (2.32GHz)
  - 单线程SGEMM
  - 性能3倍



PerfSGEMM on ARMv7

- 17层DNN模型
- Inference
  - 速度提高1倍

- 矩阵规模



总结

- DNN性能优化
  - 基于矩阵，BLAS
- OpenBLAS
  - 最好的开源BLAS实现
  - GEMM算法
- AUGEM
  - 自动生成高效汇编
  - 与手工汇编性能可以
- PerfDNN(PerfSGEMM)
  - ARMv7，提高1倍以上

谢谢