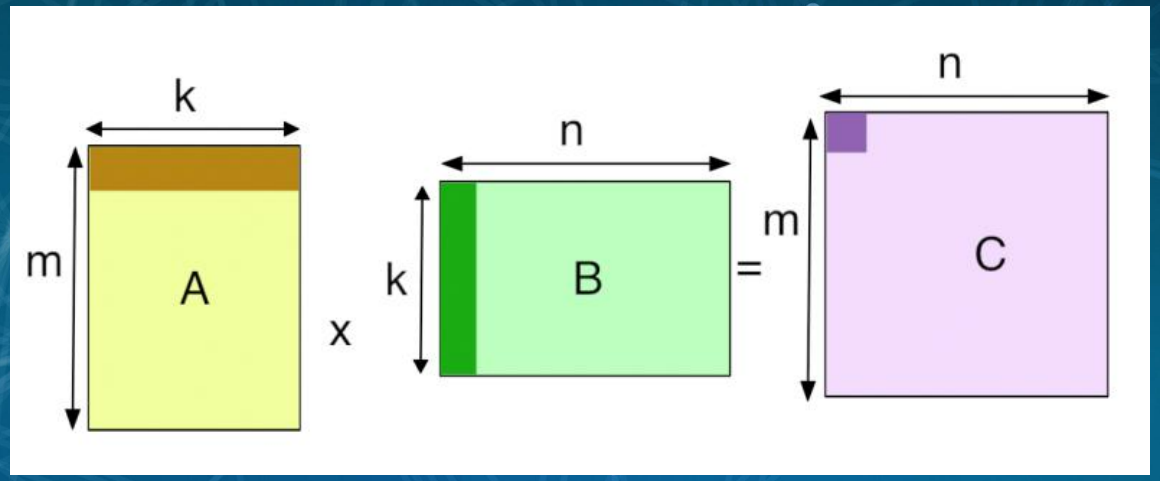
IAIS.2016
International A.I. Industry Summit
中国·光谷国际人工智能产业论坛

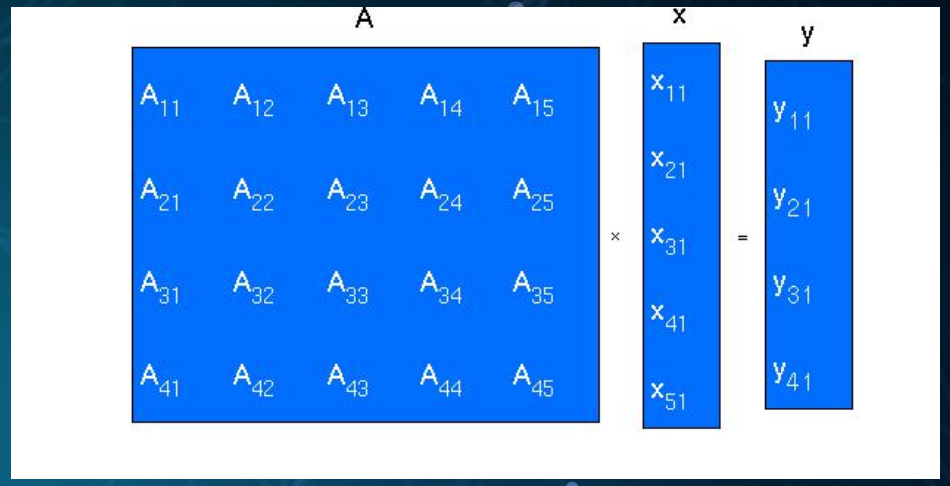
ARM上的OpenBLAS性能优化

张先轶
PerfXLab澎峰科技
xianyi@perfxlab.com

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

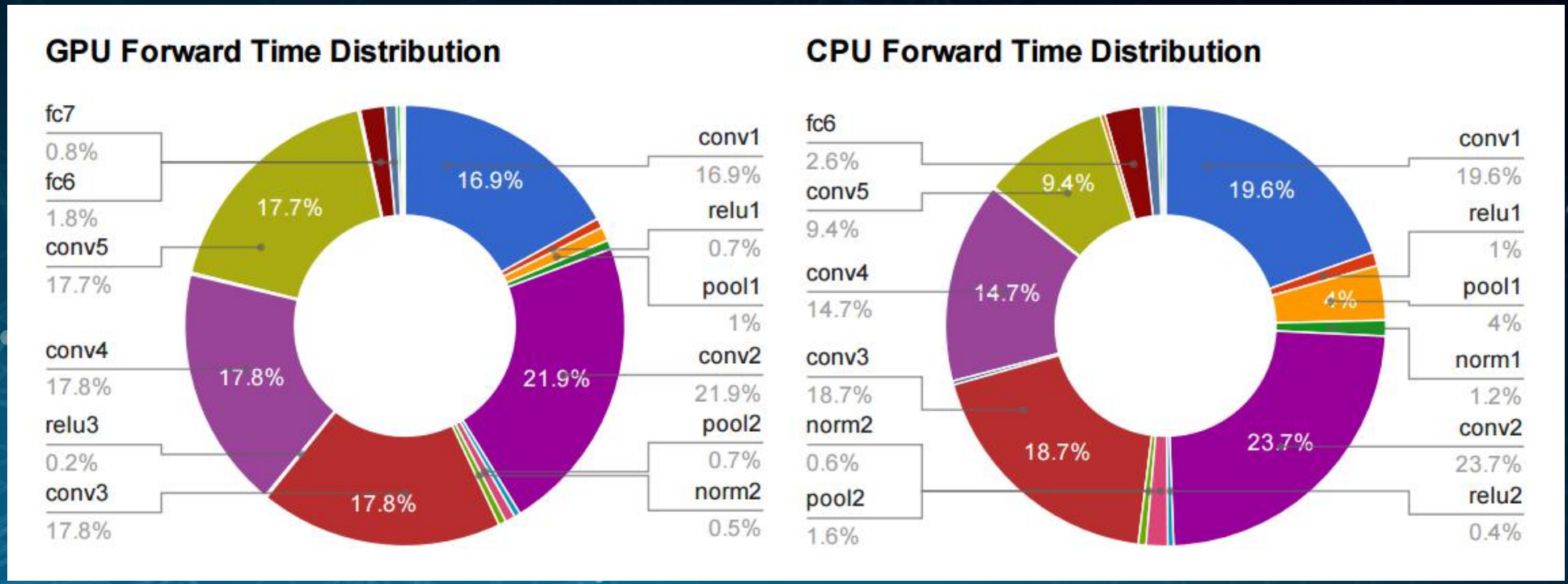




IAIS.2016
International A.I. Industry Summit
中国·光谷国际人工智能产业论坛

BLAS与深度学习

- Alexnet



IAIS.2016
International A.I. Industry Summit
中国·光谷国际人工智能产业论坛

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

IAIS.2016
International A.I. Industry Summit
中国·光谷国际人工智能产业论坛

OpenBLAS用户

julia

GNU Octave

dmlc mxnet

Caffe

Ceemple

IBM

ARM

NVIDIA

VERSATUShpc
HIGH PERFORMANCE CONSULTING

AIRBUS

PathScale
High Performance Compilers

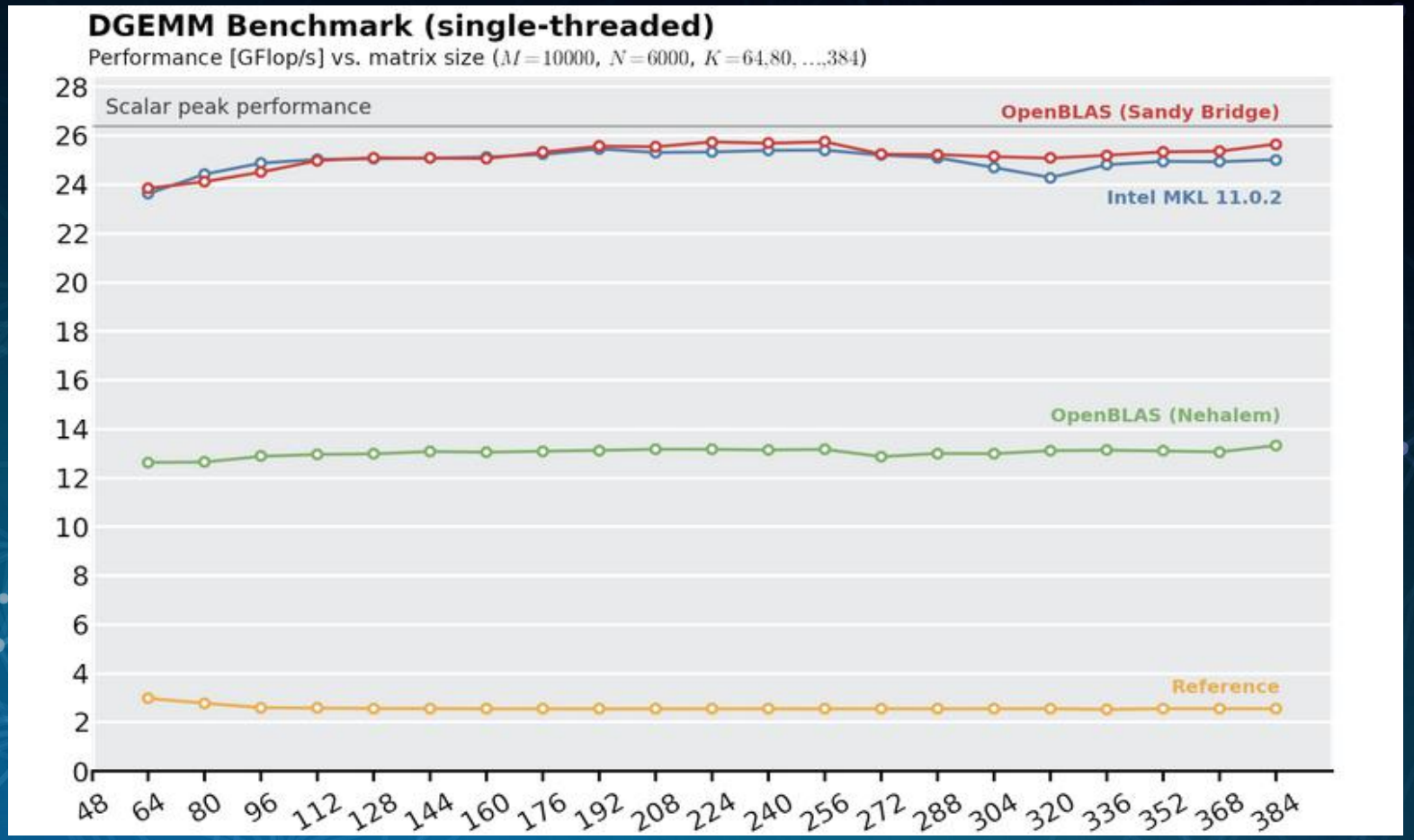
西安电子科技大学 1931 XIDIAN UNIVERSITY

Sogou搜狗

HUAQIAO UNIVERSITY 華僑大學

OpenBLAS性能

- Intel Sandy Bridge



OpenBLAS性能

- 龙芯3A

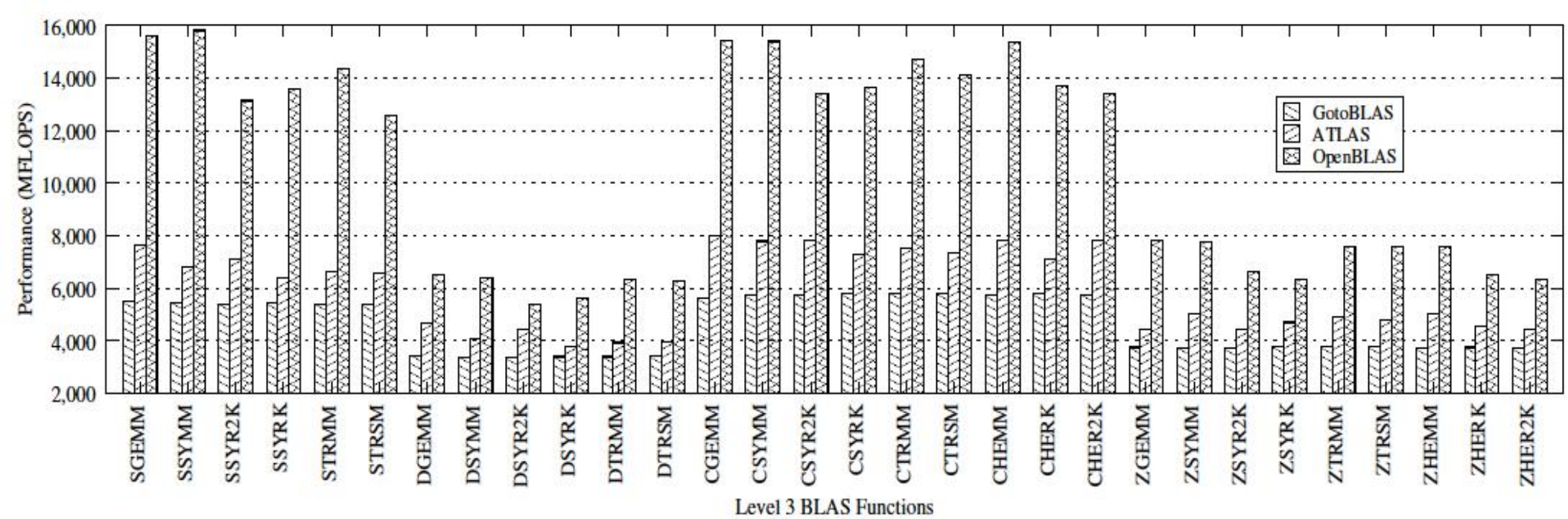
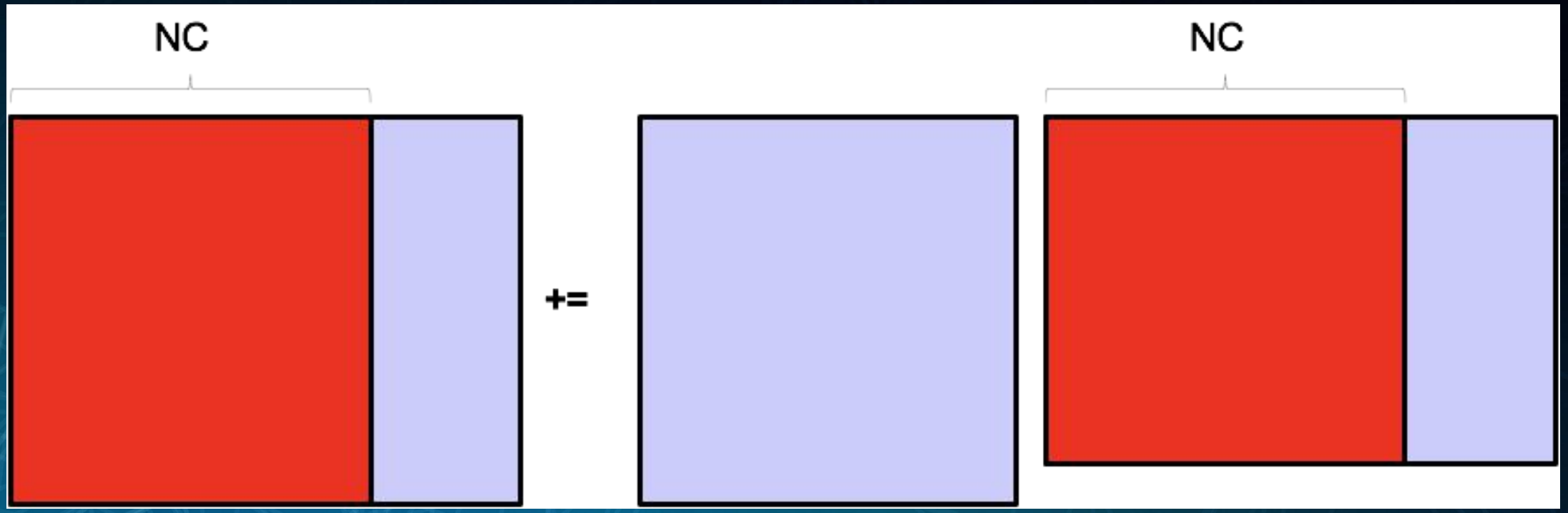


Figure 11. Multi-threaded Level 3 BLAS Performance (NP=4)

GEMM算法

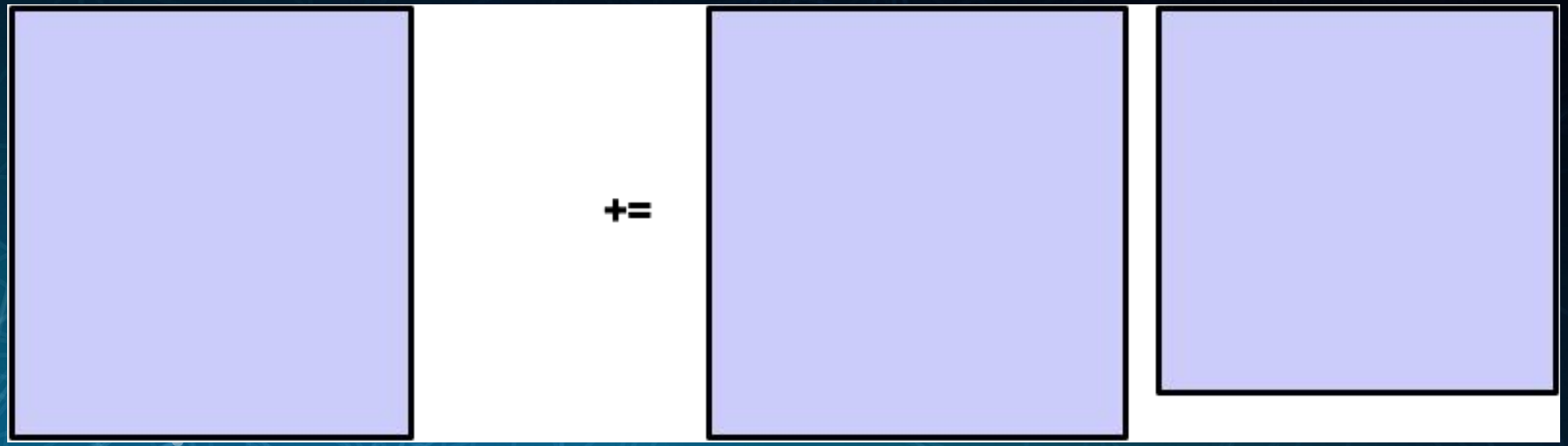
IAIS.2016
International A.I. Industry Summit
中国·光谷国际人工智能产业论坛

GEMM算法

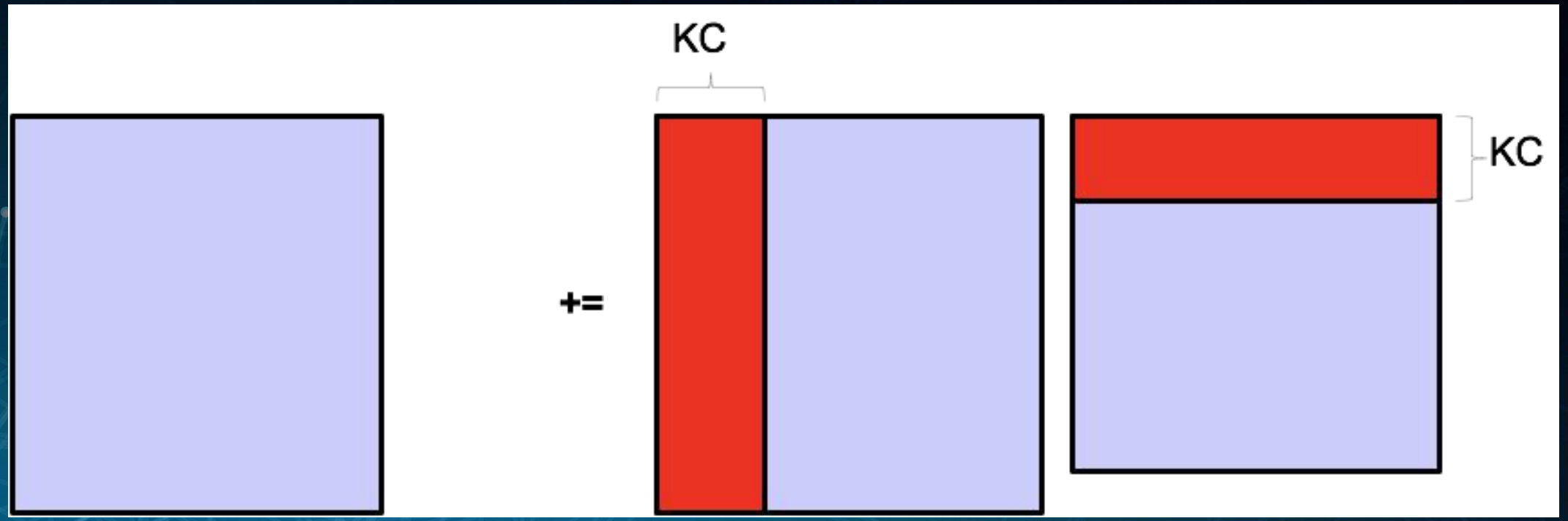


IAIS.2016
International A.I. Industry Summit
中国·光谷国际人工智能产业论坛

GEMM算法

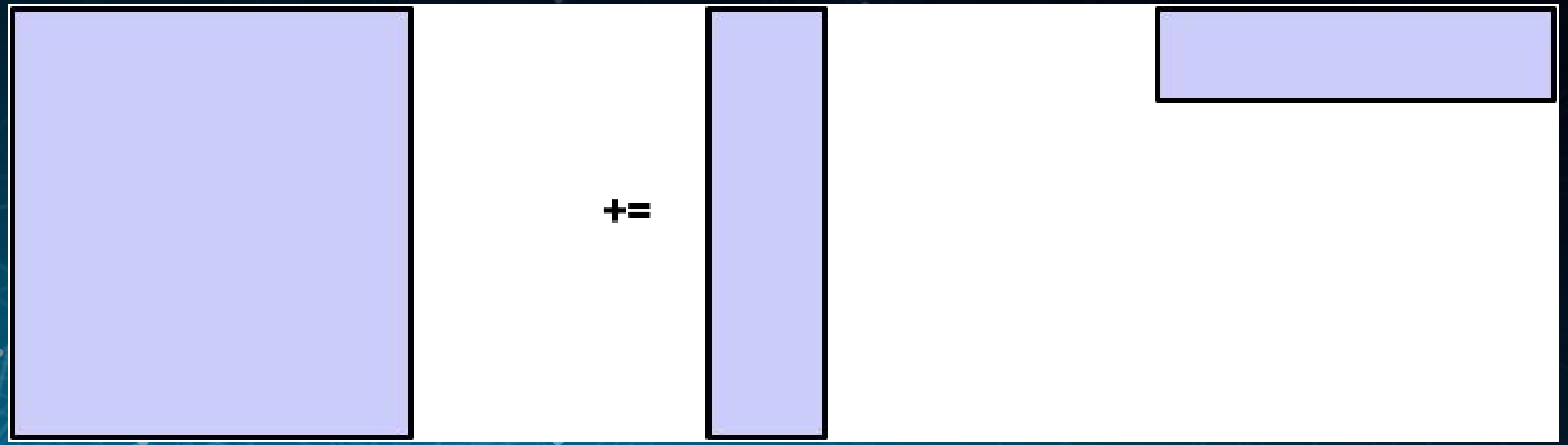


GEMM算法

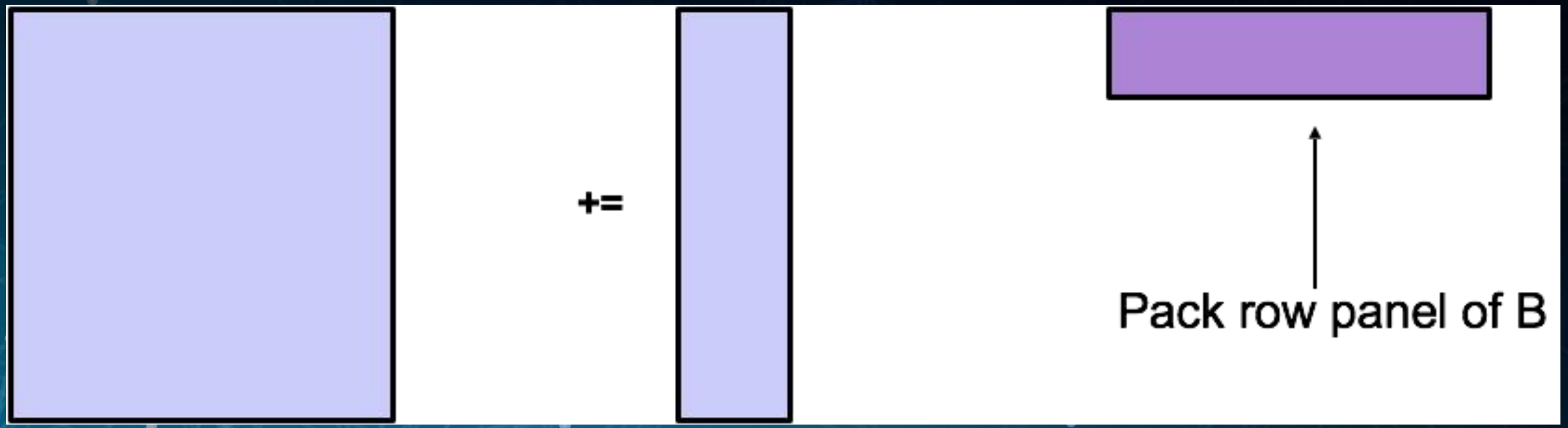


IAIS.2016
International A.I. Industry Summit
中国·光谷国际人工智能产业论坛

GEMM算法

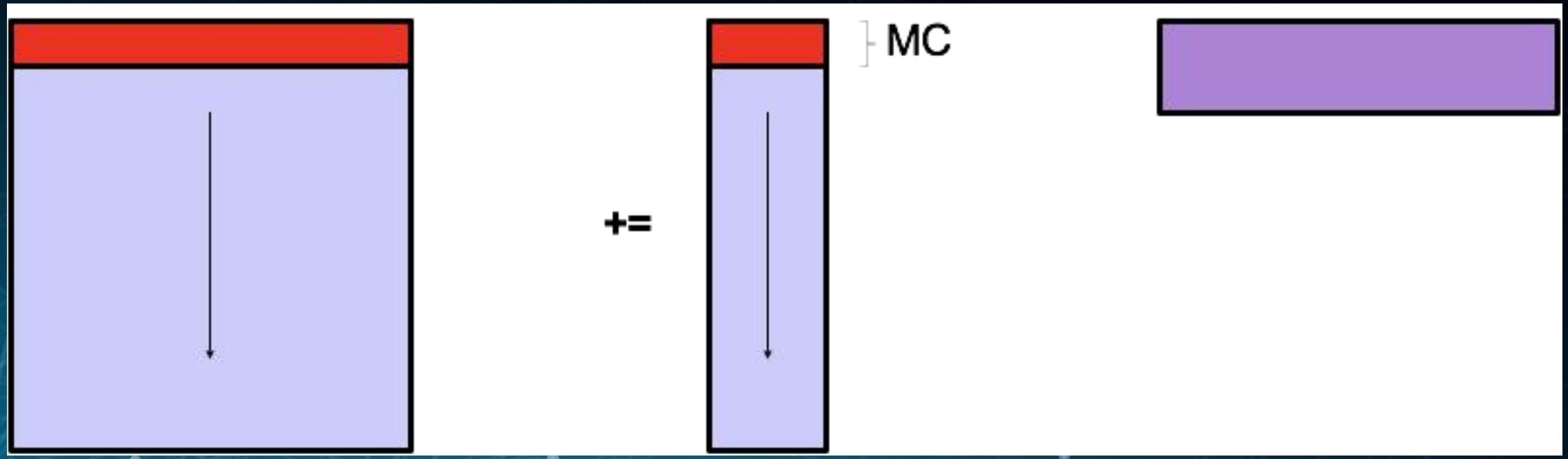


GEMM算法

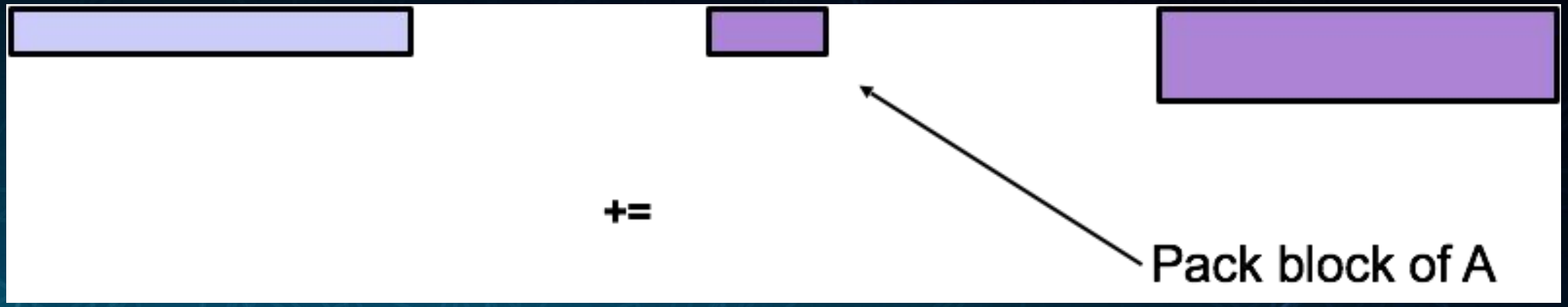


IAIS.2016
International A.I. Industry Summit
中国·光谷国际人工智能产业论坛

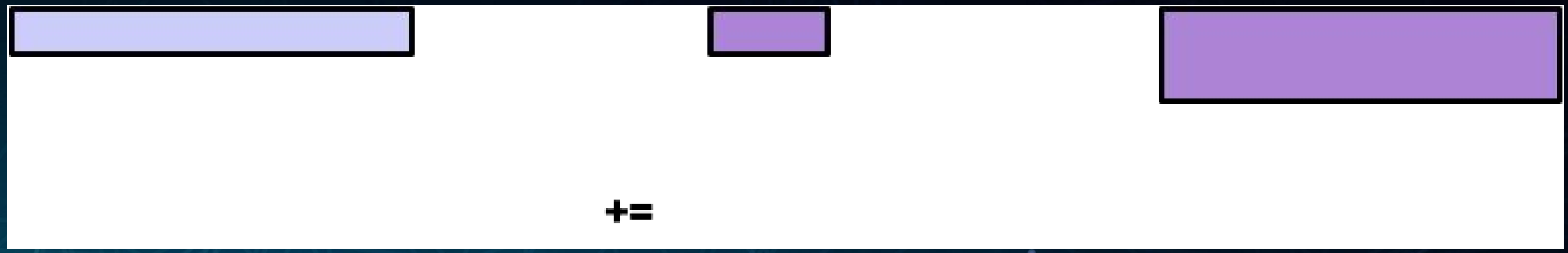
GEMM算法



GEMM算法



GEMM算法



- 核心汇编代码
  - 寄存器分块
  - 指令流水线
  - SIMD，循环展开，预取

BLAS性能优化流派

- 自动调优 Auto-tuning
  - ATLAS
  - 快速开发和移植
  - 性能一般
- 手工核心汇编
  - GotoBLAS/OpenBLAS
  - 性能好
  - 新架构？

Auto-tuning生存高效代码？？

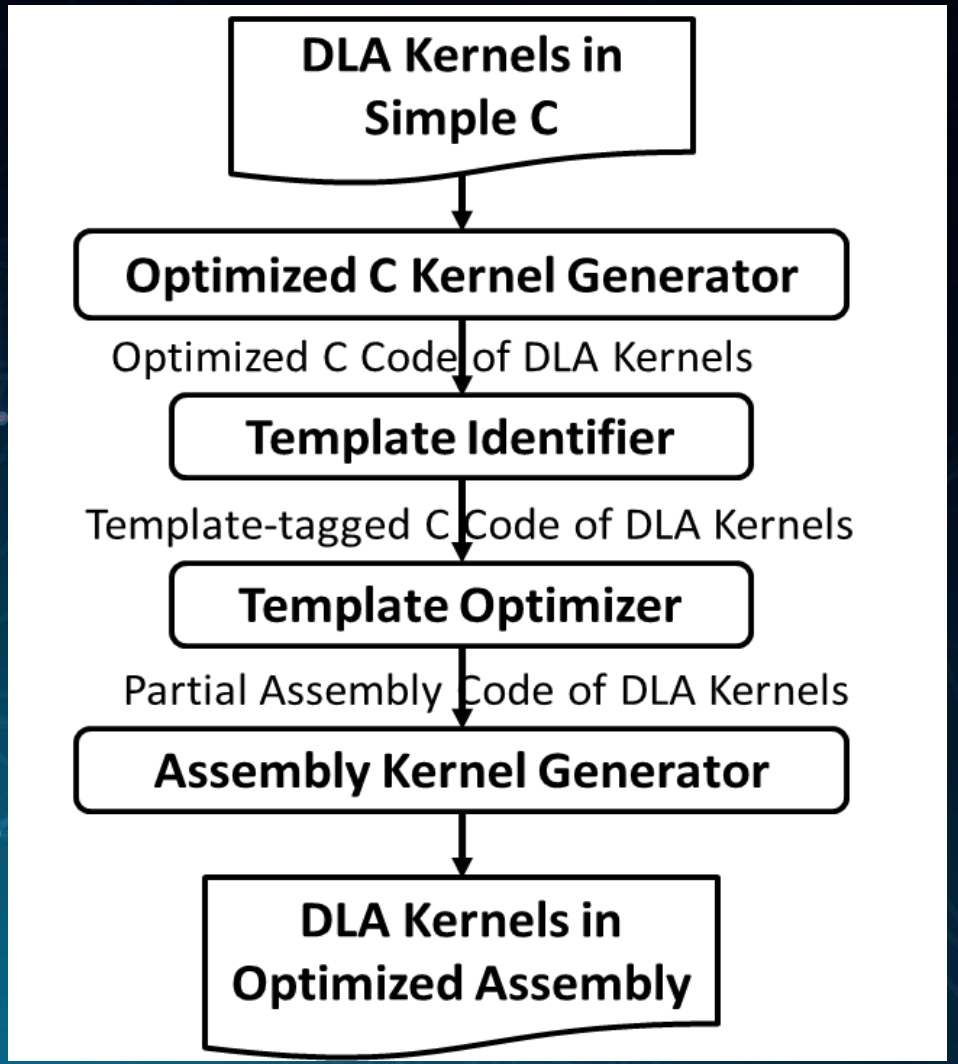
AUGEM

- 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

- 输入
  - 类似C代码片段
- 输出
  - 高性能汇编

- 基于模版Template
  - 隐含手工优化知识



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) {
1.    int i, j, l;
2.    double res, tmp1, tmp2, tmp3;
3.    for (j = 0; j < Nc; j += 1) {
4.        for (i = 0; i < Mc; i += 1) {
5.            res = 0;
6.            for (l = 0; l < Kc; l += 1) {
7.                tmp1 = A[l * Mc + i];
8.                tmp2 = B[j * Kc + l];
9.                tmp3 = tmp1 * tmp2;
10.               res = res + tmp3;
11.           }
12.           res = res * alpha;
13.           tmp1 = C[j * LDC + i];
14.           res = res + tmp1;
15.           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) {
1.   int i, j, l,...variables declaration;
2.   for (j = 0; j < Nc;  j += 2) {
3.       ptr_A = A; ptr_C0 = C; ptr_C1 = C + LDC;
4.       pre_B = B + 2 * Kc;
5.       for (i = 0; I < MC; I += 2) {
6.           Prefetch(pre_B); ptr_B = B;
7.           res0 = 0; res1 = 0; res2 = 0; res3 = 0;
8.           for (l = 0; l < Kc; l += 4) {
9.               Prefetch(ptr_A+PREDIS);
10.              tmp0 = ptr_A[0];
11.              tmp1 = ptr_B[0];
12.              tmp2 = tmp0 * tmp1;
13.              res0 = res0 + tmp2;
14.              tmp0 = ptr_A[1];
15.              tmp1 = ptr_B[0];
16.              tmp2 = tmp0 * tmp1;
17.              res1 = res1 + tmp2;
18.              ...
19.              ptr_A += 4; ptr_B += 4;
20.          } ...cleanup code for loop l
21.          res0 = res0 * alpha; res1 = res1 * alpha; ...
22.          tmp0 = C[j * LDC +i];
23.          res0 = res0 + tmp0;
24.          C[j * LDC + i] = res0;
25.          ...
26.      }...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) {
1.  int i, j, l,...variables declaration;
2.  for (j = 0; j < Nc;  j += 2) {
3.      ptr_A = A; ptr_C0 = C; ptr_C1 = C + LDC;
4.      pre_B = B + 2 * Kc;
5.      for (i = 0; I < MC; I += 2) {
6.          Prefetch(pre_B); ptr_B = B;
7.          res0 = 0; res1 = 0; res2 = 0; res3 = 0;
8.          for (l = 0; l < Kc; l += 4) {
9.              Prefetch(ptr_A+PREDIS);
10.             tmp0 = ptr_A[0];
11.             tmp1 = ptr_B[0];
12.             tmp2 = tmp0 * tmp1;
13.             res0 = res0 + tmp2;
14.             tmp0 = ptr_A[1];
15.             tmp1 = ptr_B[0];
16.             tmp2 = tmp0 * tmp1;
17.             res1 = res1 + tmp2;
18.             ...
19.             ptr_A += 4; ptr_B += 4;
20.         } ...cleanup code for loop l
21.         res0 = res0 * alpha; res1 = res1 * alpha; ...
22.         tmp0 = C[j * LDC +i];
23.         res0 = res0 + tmp0;
24.         C[j * LDC + i] = res0;
25.         ...
26.     }...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Template
  - 2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;	**mmUnrolledSTOR**E(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,res1,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
    - B
    - C
    - 中间结果
  - 不考虑寄存器溢出
    - 临时保存到堆栈 X

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

Template优化

- 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

- 将剩余代码生成汇编
  - 循环控制
- 保持寄存器分配的一致性
  - 引入reg_table
  - 全局记录表

Algorithm of Template Optimizer

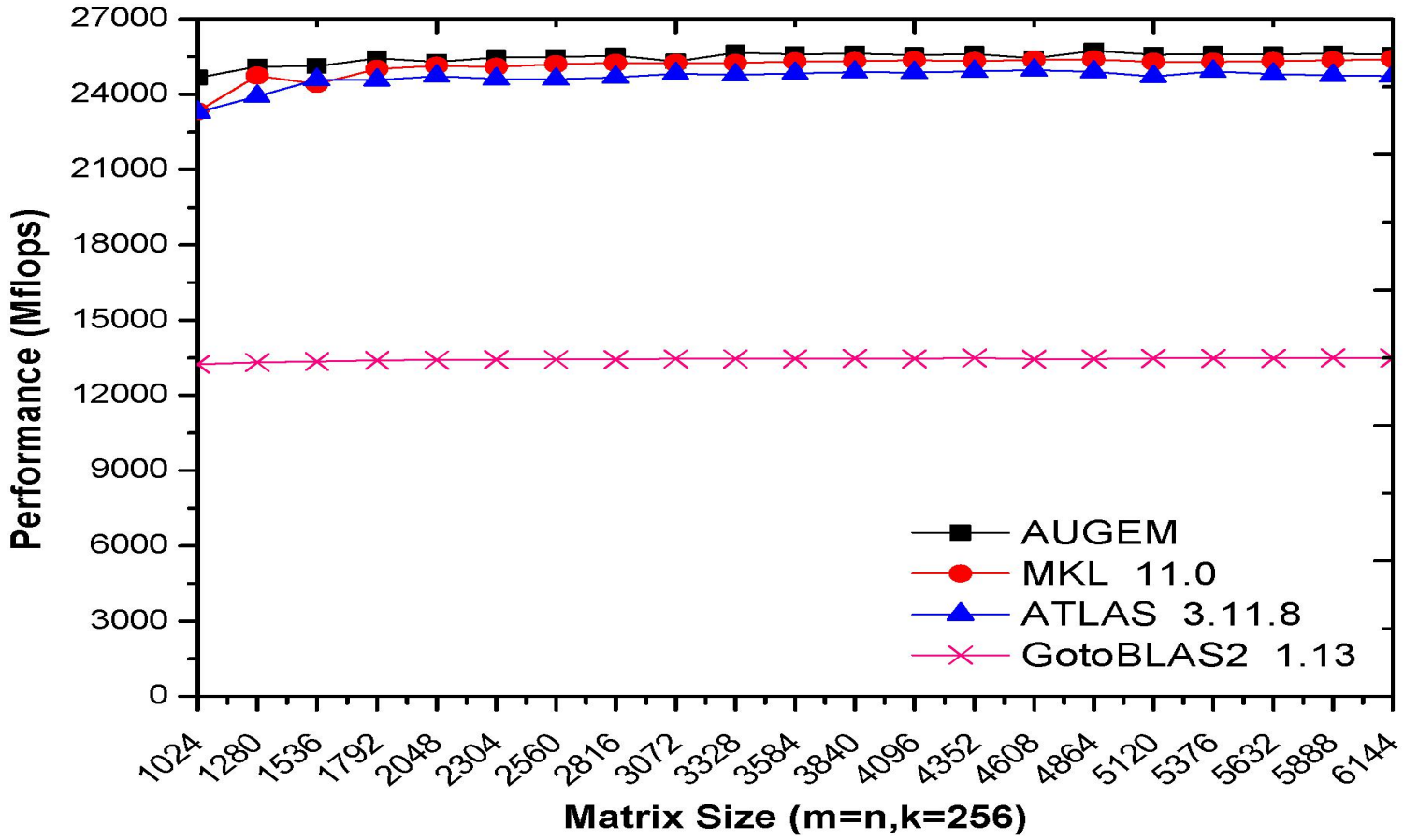
Input: *input*: template-annotated kernel in low-level C
arch: architecture specification
Output: *res*: optimized kernel in assembly
1: $res = input$;
2: reg_table = empty;
3: reg_free=available_registers($arch$);
4: **for** each annotated code region r in *input* **do**
5: r_annot = template_annotation(r);
6: $r1$ = Optimizer[r_annot]
(r, reg_table, reg_free, $arch$);
7: res = replace $r1$ with r in res;
8: **end for**

AUGEM性能测试

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

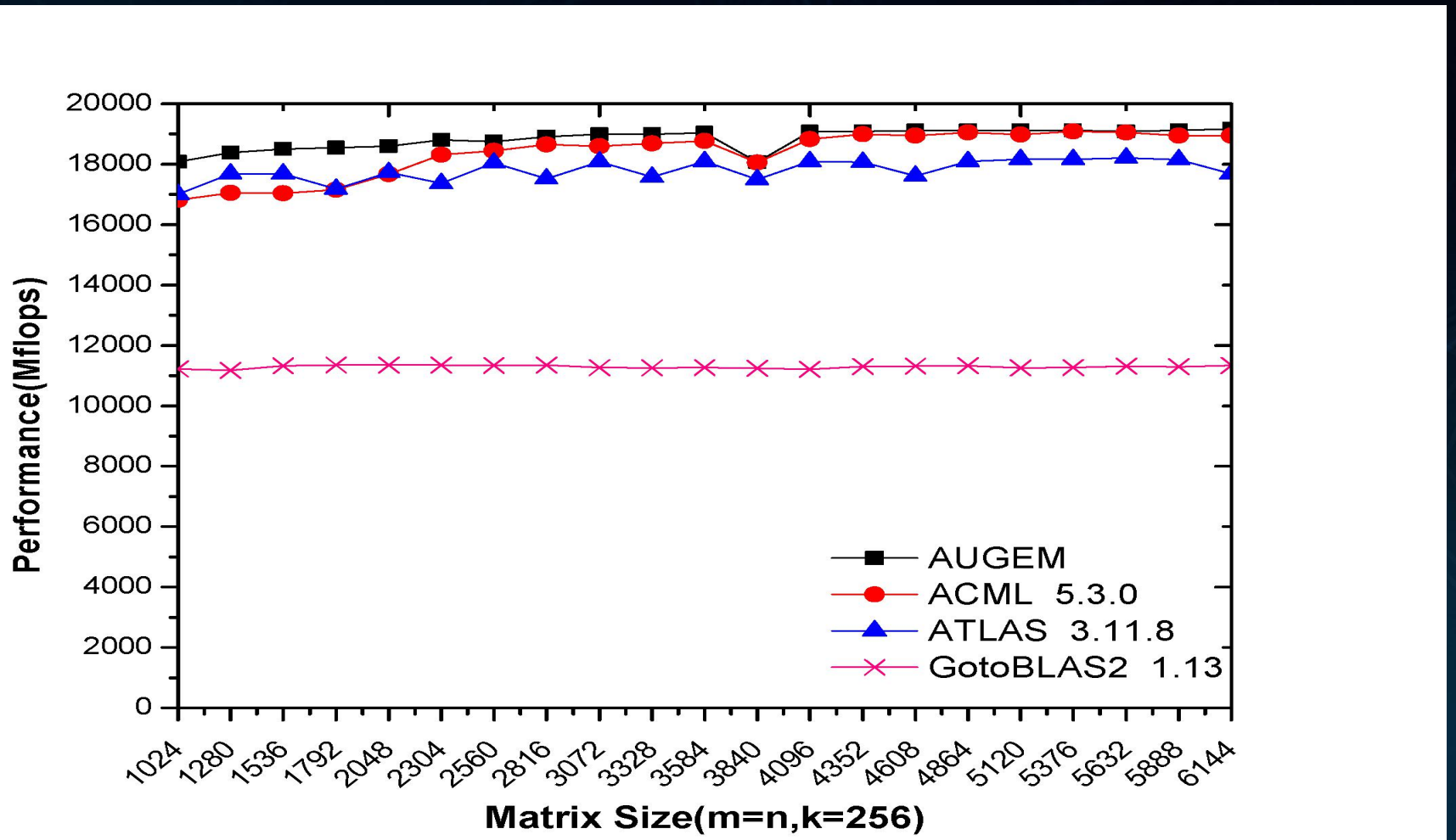
AUGEM性能测试

- DGEMM on Intel Sandy Bridge



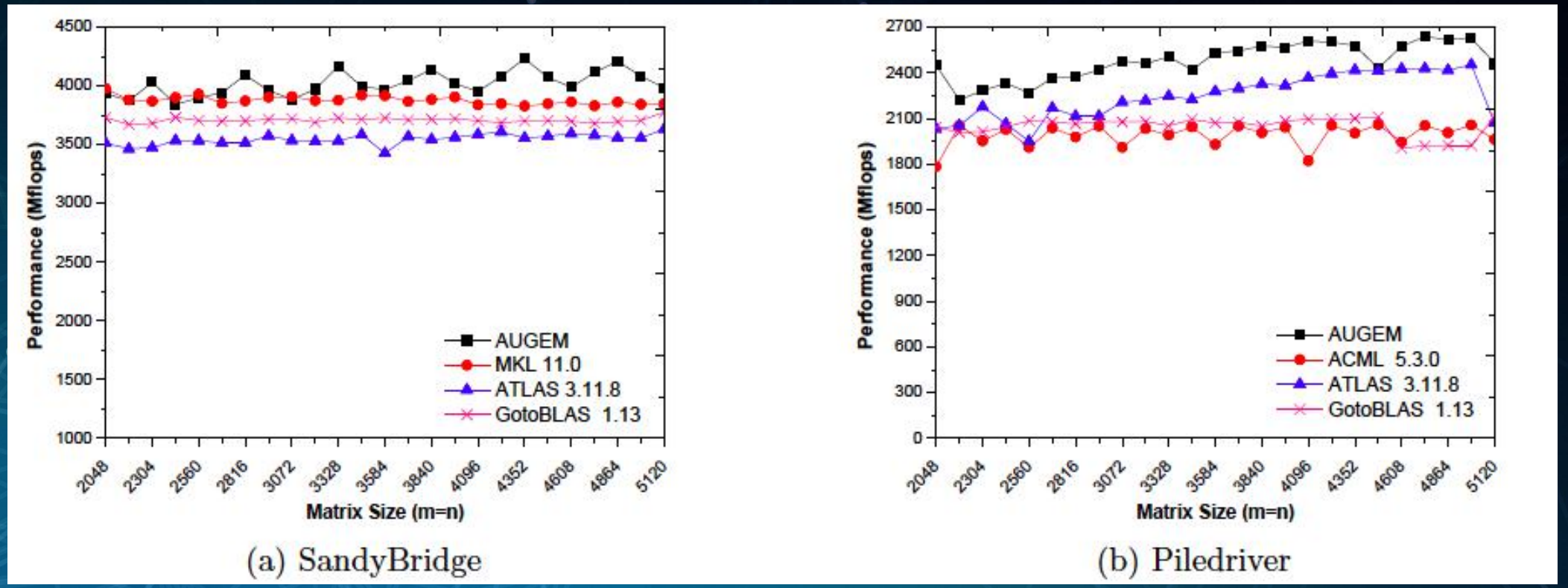
AUGEM性能测试

- DGEMM on AMD Piledriver



AUGEM性能测试

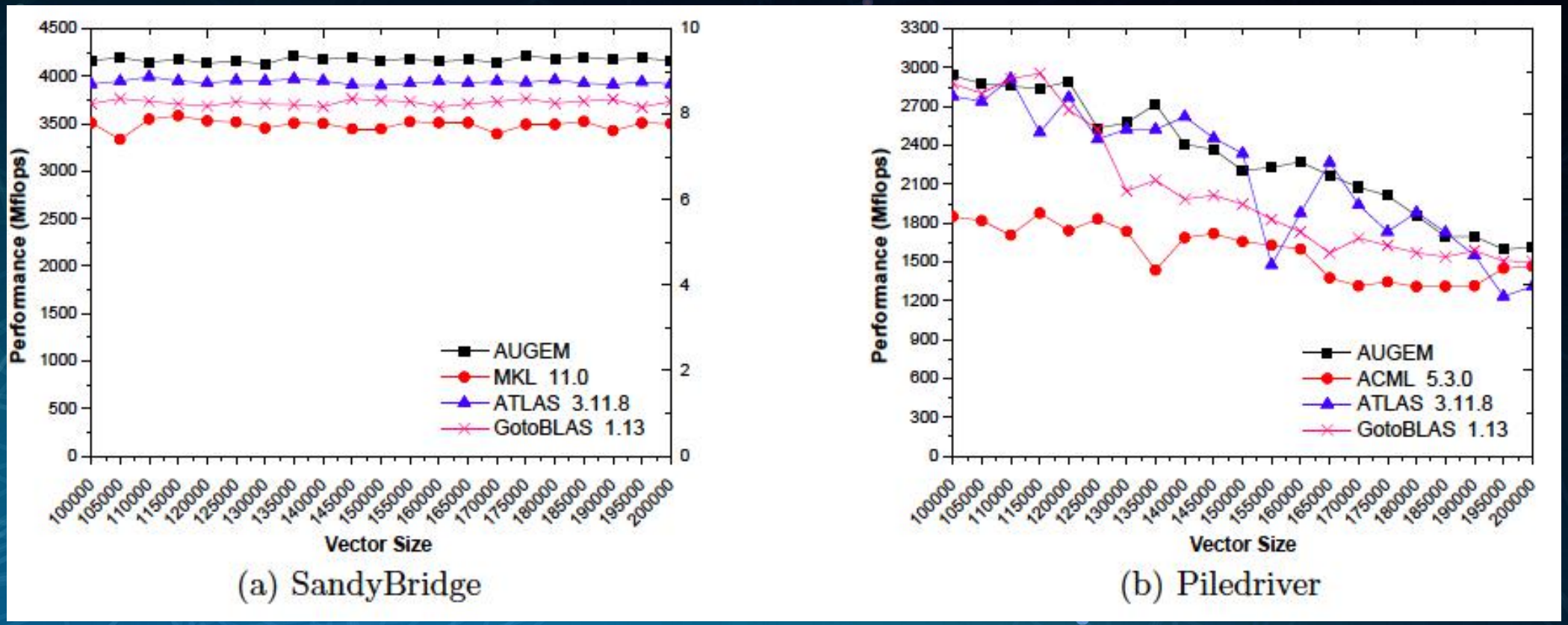
- DGEMV（BLAS2级）



IAIS.2016
International A.I. Industry Summit
中国·光谷国际人工智能产业论坛

AUGEM性能测试

- DAXPY (BLAS1级)

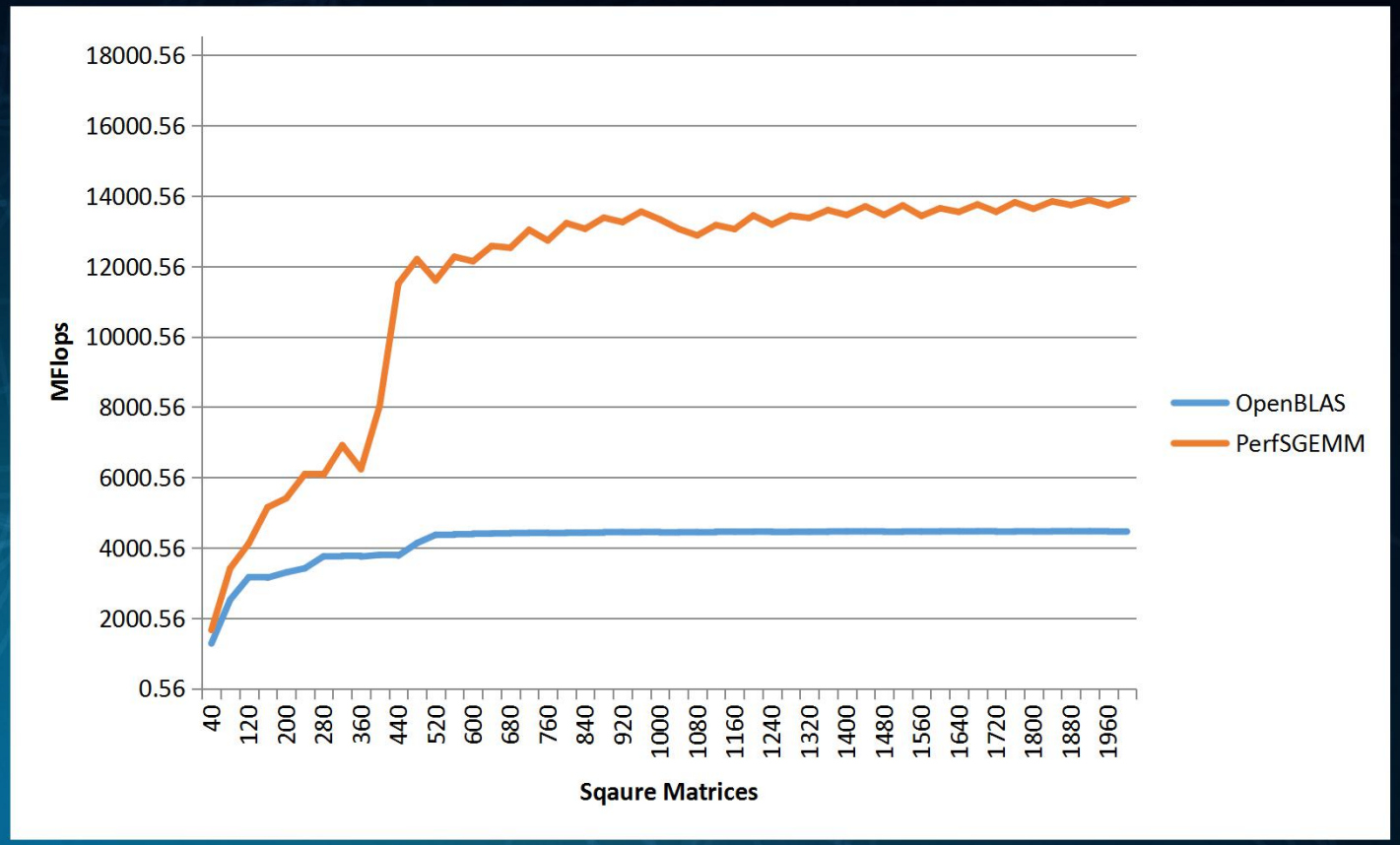


PerfSGEMM on ARM

- 单精度SGEMM
  - Neon SIMD指令
  - 与IEEE 754标准不一致
  - Round mode (Flush to Zero)
  - 不影响深度学习的精度
- AUGEM生成汇编代码
  - 支持ARM特殊指令
    - vmla.f32 c, a, b[d]

PerfSGEMM on ARM

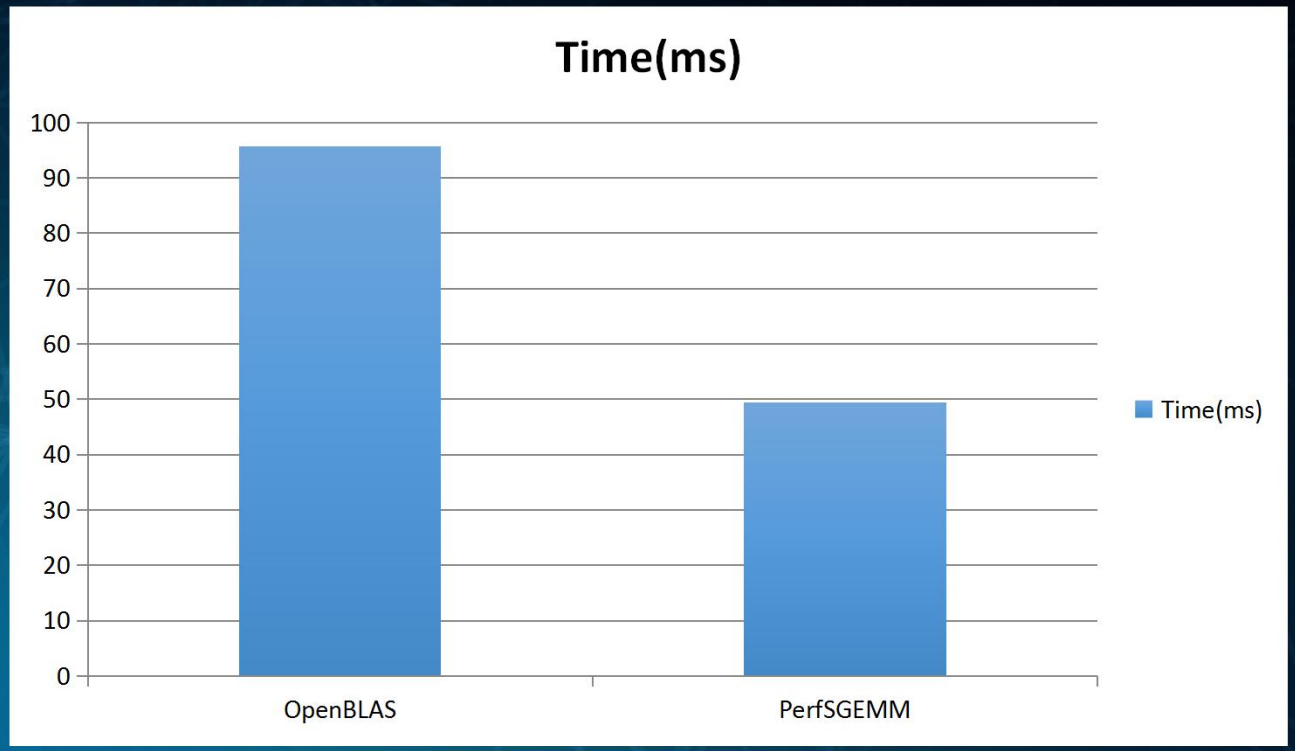
- ARM Cortex A15 (2.32GHz)
  - 单线程SGEMM
  - 性能3倍



IAIS.2016
International A.I. Industry Summit
中国·光谷国际人工智能产业论坛

PerfSGEMM on ARM

- 17层DNN模型
- Inference
  - 速度提高1倍



总结

- OpenBLAS
  - 最好的开源BLAS实现
- AUGEM
  - 自动生成高效汇编
  - 与手工汇编性能可比
- PerfSGEMM
  - ARMv7
  - 深度学习

IAIS.2016
International A.I. Industry Summit
中国·光谷国际人工智能产业论坛

谢谢

xianyi@perfxlab.com